

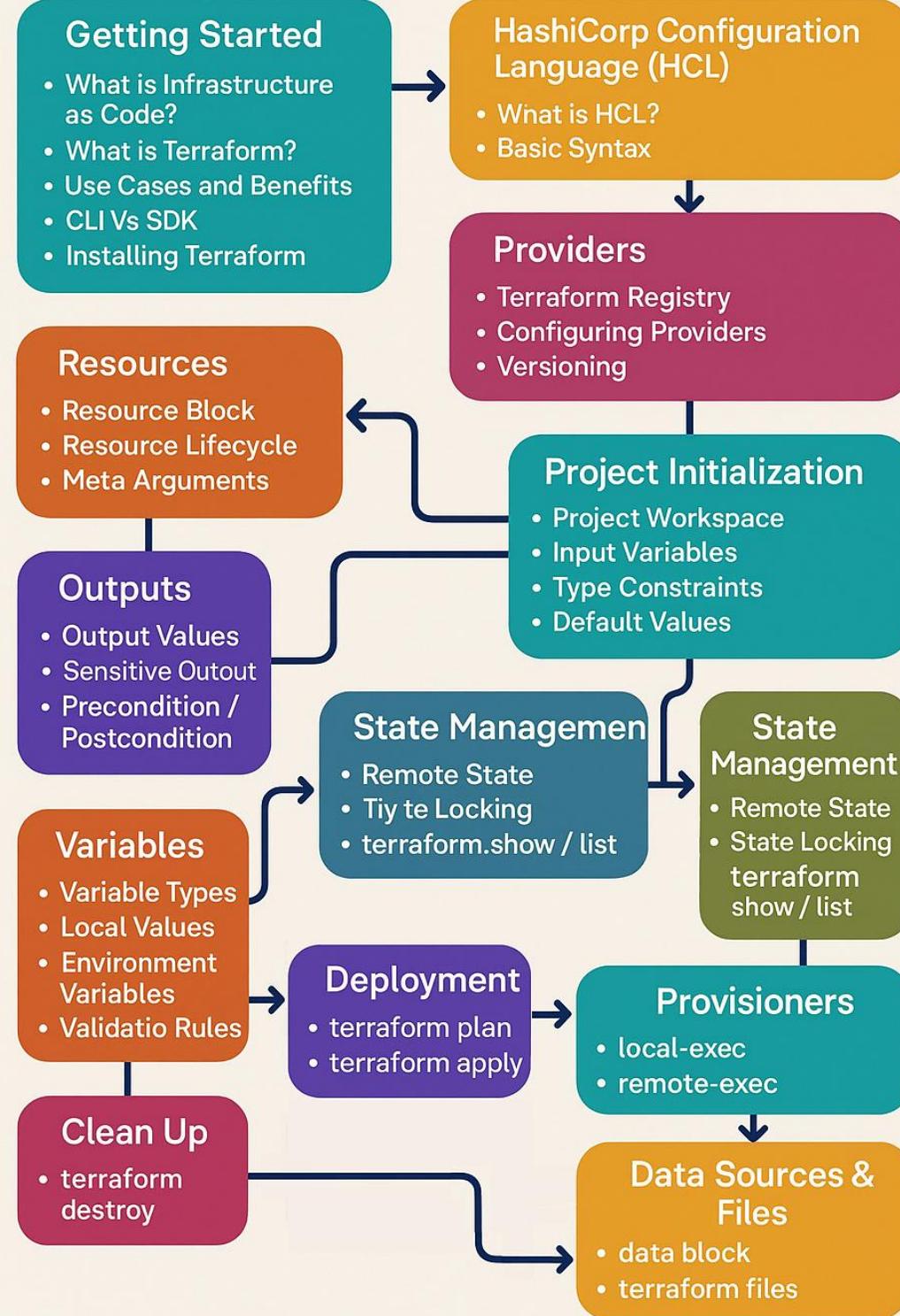
# Hashicorp Terraform

**Saleh Miri**

- Phone: 09126117464
- Email: [salehmiri90@gmail.com](mailto:salehmiri90@gmail.com)
- [Linkedin.com/in/salehmiri](https://www.linkedin.com/in/salehmiri)
- [Github.com/salehmiri90](https://github.com/salehmiri90)
- [Youtube.com/salehmiri90](https://youtube.com/salehmiri90)

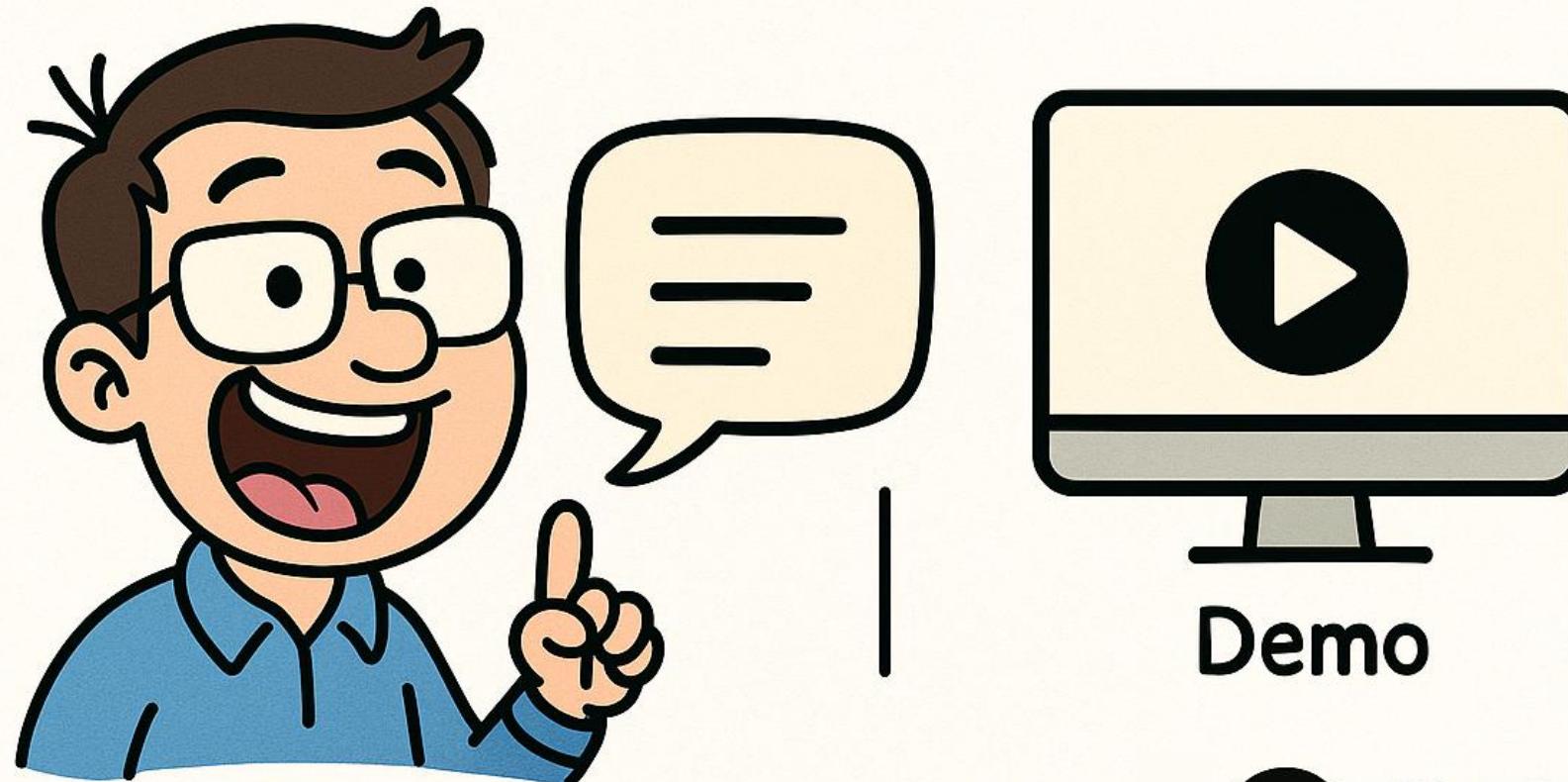


# Mastering Terraform



# Teaching Terraform

combination of explanation and demo



with step-by-step guide on  GitHub

[Code](#) [Issues](#) [Pull requests](#) [Actions](#) [Projects](#) [Security](#) [Insights](#)

main terraform-course / 04-Terraform-Resources /

salehmiri90 add readme 04-06

# GitHub Step-by-Step Documentation

Name	Last commit message
..	
04-01-Resource-Syntax-and-Behavior	edit instance hardware
04-02-Meta-Arguments	edit instance hardware
04-03-Meta-Argument-count	edit instance hardware

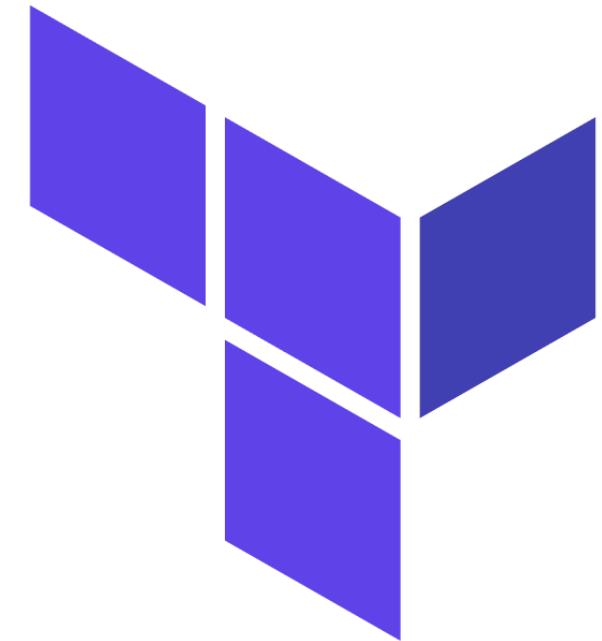
# Practical Examples For Each Concept

- ✓ Terraform-Course-1404
  - ✓ 01-Infrastructure-as-Code-iaC
    - (i) README.md
  - ✓ 02-Terraform-Basics
    - > 02-01-Install-Tools-TerraformCLI-VSCodeIDE
    - > 02-02-Terraform-Command-Basics
    - > 02-03-Terraform-Language-Syntax
  - ✓ 03-Terraform-Fundamental-Blocks
    - > 03-01-Terraform-Block
    - > 03-02-Provider-Block
    - > 03-03-Multiple-Provider-Configurations
    - > 03-04-Providers-Dependency-Lock-File
  - ✓ 04-Terraform-Resources
    - > 04-01-Resource-Syntax-and-Behavior
    - > 04-02-Meta-Arguments
    - > 04-03-Meta-Argument-count
    - > 04-04-Meta-Argument-for\_each
    - > 04-05-Meta-Argument-lifecycle
    - > 04-06-Provisioners
  - ✓ 05-Terraform-Variables
    - > 05-01-Terraform-Input-Variables
    - > 05-02-Terraform-Output-Values
    - > 05-03-Terraform-Local-Values

- ✓ 04-04-Meta-Argument-for\_each
  - ✓ v1-for\_each-maps
    - ✓ c1-versions.tf
    - ✓ c2-s3bucket.tf
  - ✓ v2-for\_each-toset
    - ✓ c1-versions.tf
    - ✓ c2-iamuser.tf
  - (i) README.md
- ✓ 04-05-Meta-Argument-lifecycle
  - ✓ v1-create\_before\_destroy
    - ✓ c1-versions.tf
    - ✓ c2-ec2-instance.tf
  - ✓ v2-prevent\_destroy
    - ✓ c1-versions.tf
    - ✓ c2-ec2-instance.tf
  - ✓ v3-ignore\_changes
    - ✓ c1-versions.tf
    - ✓ c2-ec2-instance.tf

# Introduction

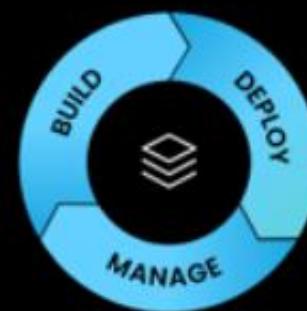
- HashiCorp Products?
- What is Terraform?
- Is it worth to learn Terraform in 2025?
- Prerequisite for course



# HashiCorp Products

## The Infrastructure Cloud

### Infrastructure Lifecycle Management



A circular diagram representing the Infrastructure Lifecycle Management process. It consists of three main phases: BUILD (top), DEPLOY (right), and MANAGE (bottom). The phases are connected by arrows forming a continuous loop.

### Security Lifecycle Management



A circular diagram representing the Security Lifecycle Management process. It consists of three main phases: CONNECT (top), PROTECT (right), and INSPECT (bottom). The phases are connected by arrows forming a continuous loop.

#### Platform Capabilities

Automation & Optimization



Terraform

Visibility & Policy



Waypoint

Reliability & Scale



Packer

Integrations & API



Vault

Governance, Risk, Compliance



Boundary

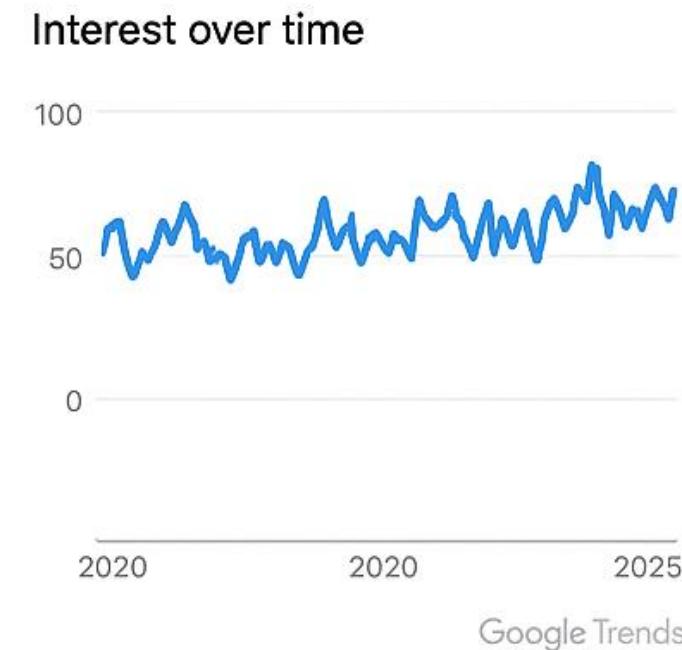


Consul



# Is it worth to learn Terraform in 2025?

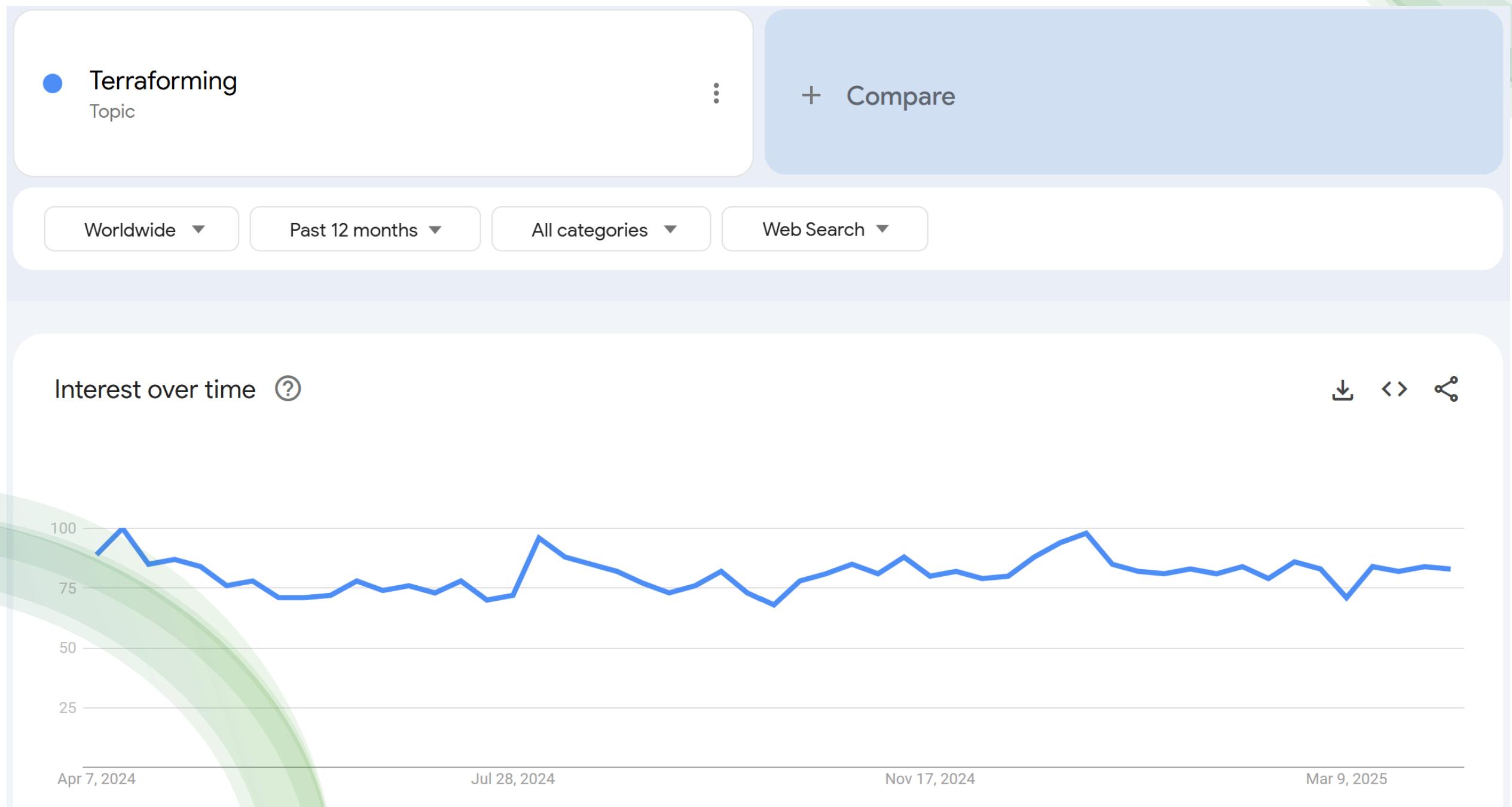
 Yes, learning Terraform remains a key and popular skill in IT and DevOps



## Current Status of Terraform In 2025:

- High popularity: Terraform is still one of the most widely used IaC tools (IaC) tools
- Broad adoption: Many companies and organizations continue to use and invest in Terraform
- Ongoing development: New features and improvements are being regularly released

# Google Trends – Past 1 Year

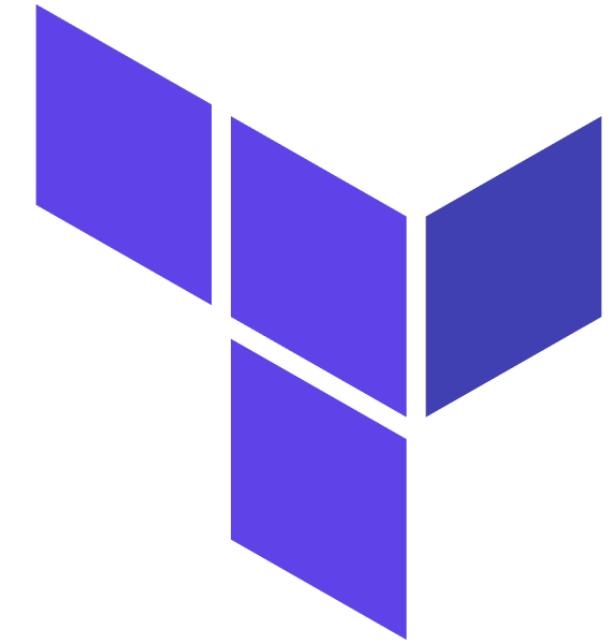


# Prerequisite for this course?



# 01: Infrastructure as Code (IaC)

- **Infrastructure as Code?**
- **Traditional Way of Managing Infrastructure**
- **Manage using IaC with Terraform**
- **Terraform placement in Infrastructure**
- **Declarative vs Procedural approaches**
- **Pull vs Push execution methods**
- **Terraform vs Ansible**
- **Terraform Website and Providers**



# Infrastructure as Code?

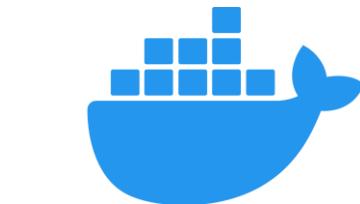
Configuration Management



ANSIBLE



Server Templating



docker

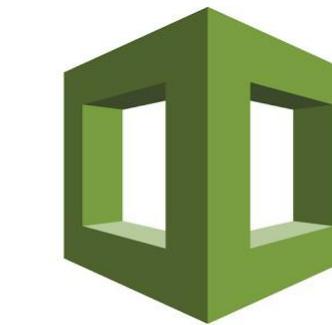


HashiCorp  
Vagrant



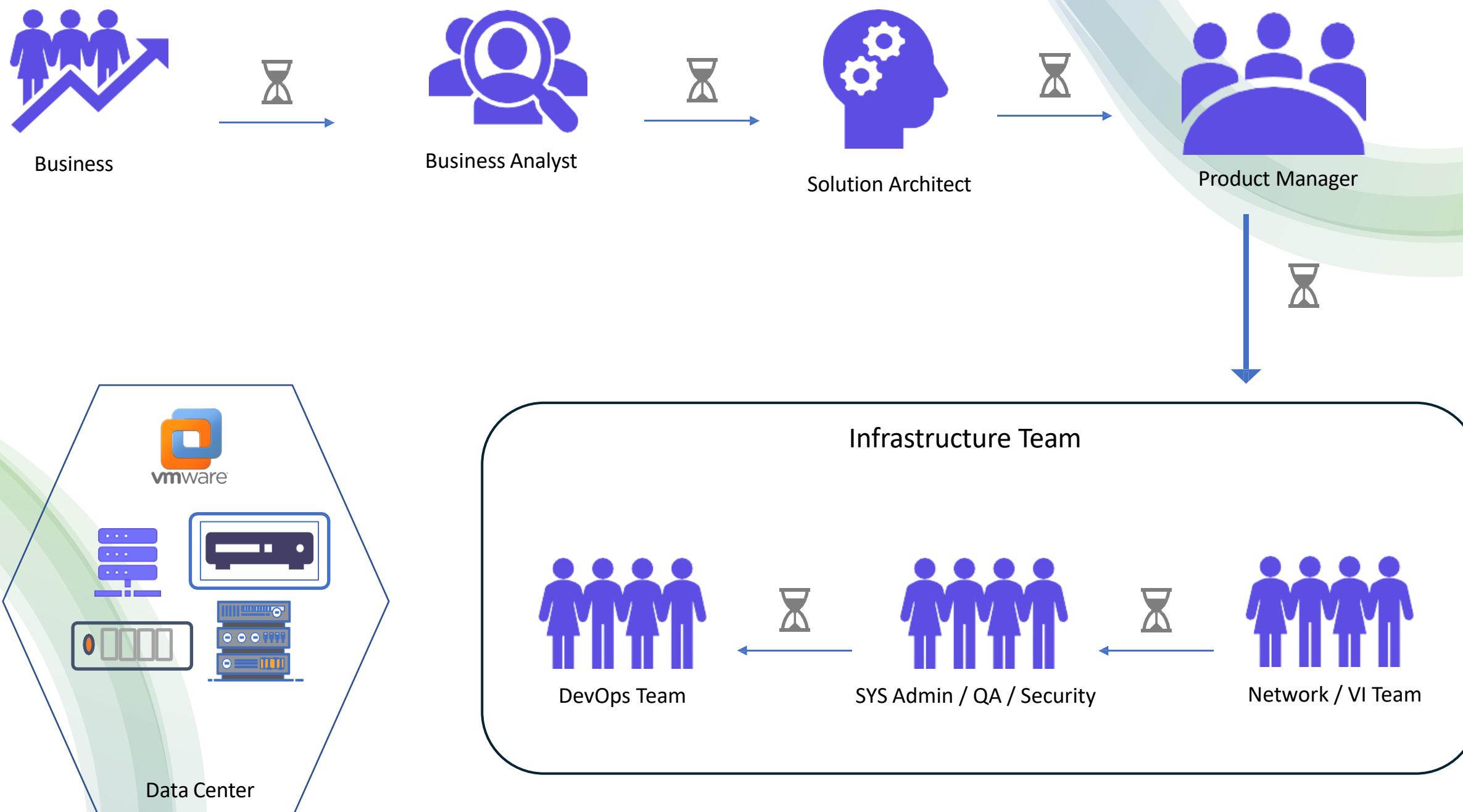
HashiCorp  
Packer

Provisioning Tools

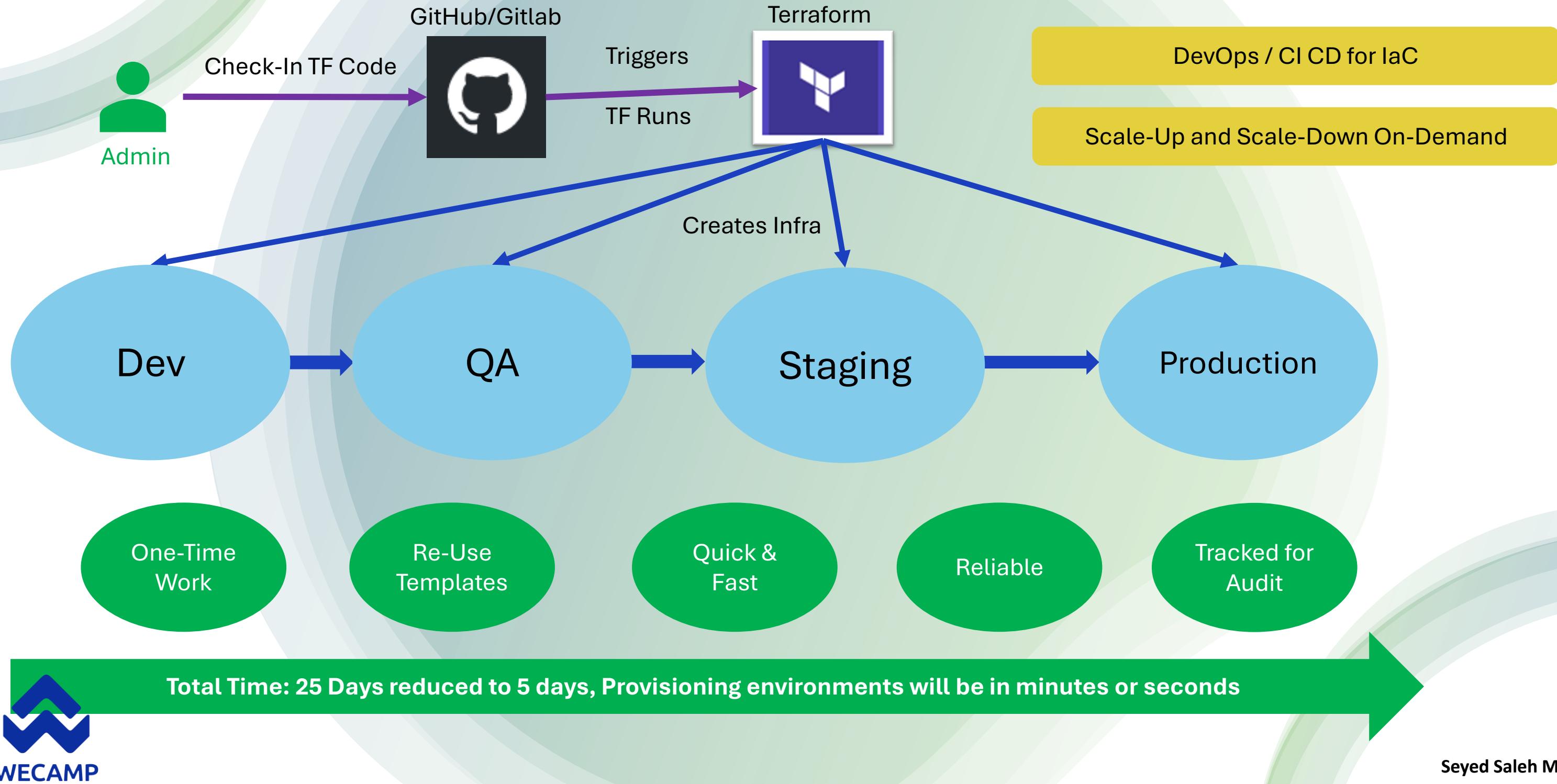


CloudFormation

# Traditional Way of Managing Infrastructure



# Manage using IaC with Terraform



# Manage using IaC with Terraform

## Visibility

IaC serves as a very **clear reference** of what resources we created, and what their settings are. We don't have to **navigate** to the web console to check the parameters.

## Stability

If you **accidentally** change the **wrong** setting or delete the **wrong** resource in the web console you can **break things**. IaC helps **solve** this, especially when it is combined with **version control**, such as Git.

## Scalability

With IaC we can **write it once** and then **reuse it many times**. This means that one well written template can be used as the **basis** for multiple services, in multiple regions around the world, making it much easier to horizontally scale.

## Security

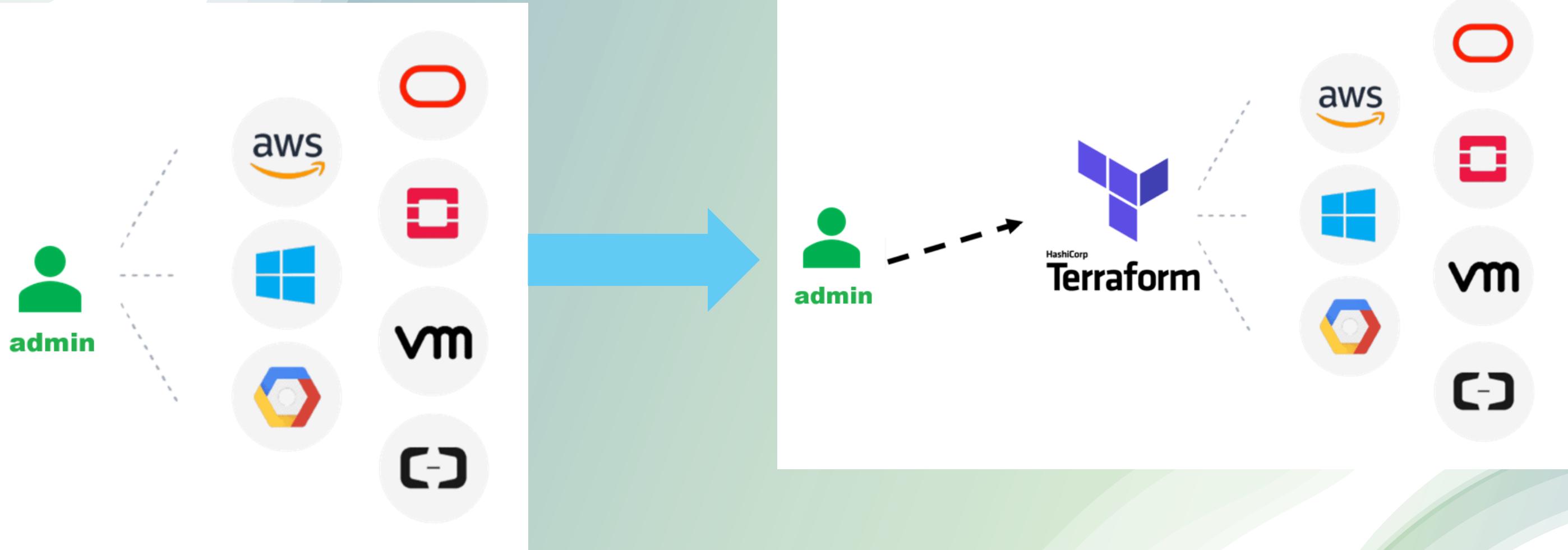
Once again IaC gives you a **unified template** for how to deploy our architecture. If we create one well **secured architecture** we can reuse it multiple times, and know that each deployed version is following the same settings.

## Audit

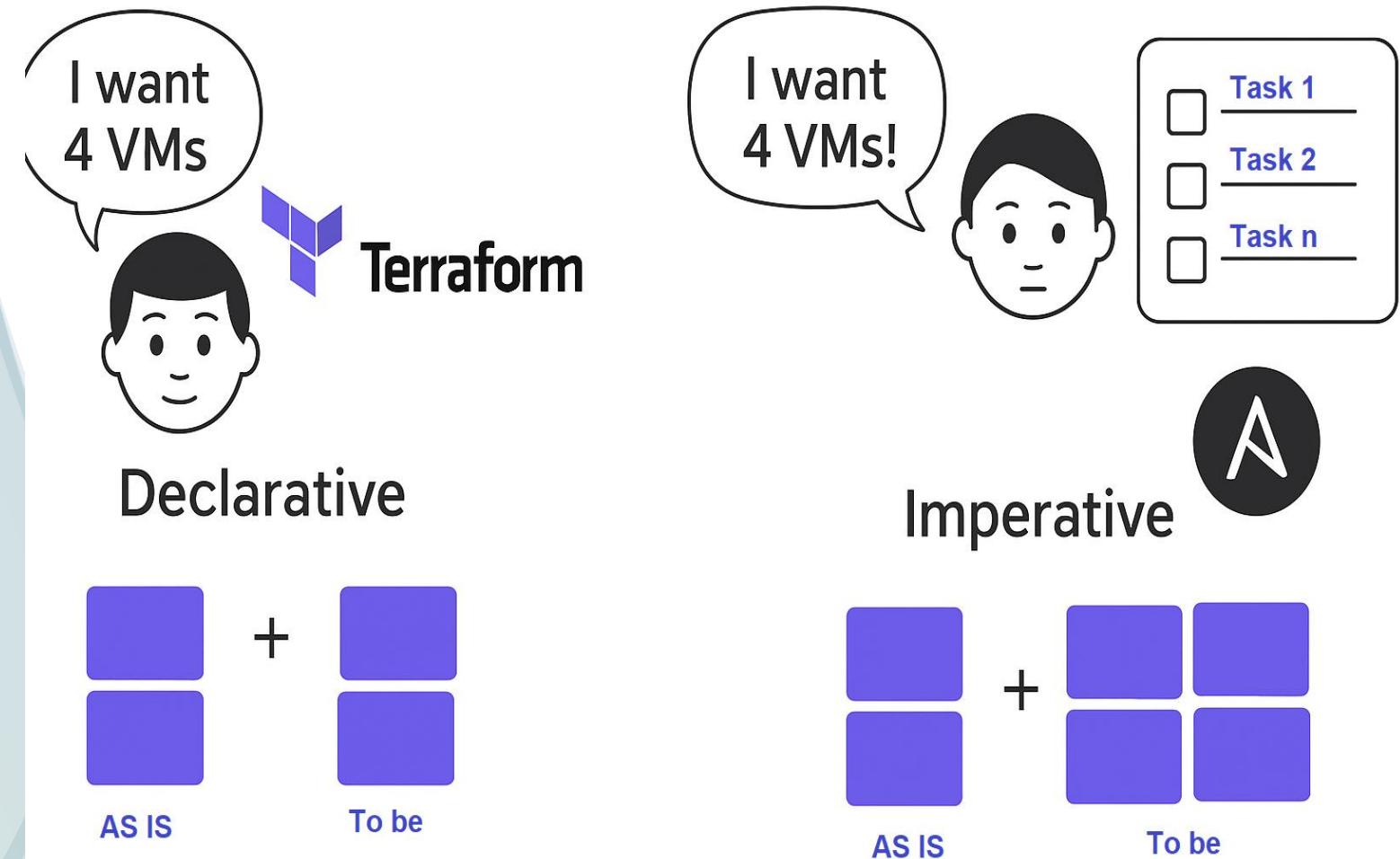
Terraform not only creates resources it also **maintains the record** of what is created in real world cloud environments using its State files.



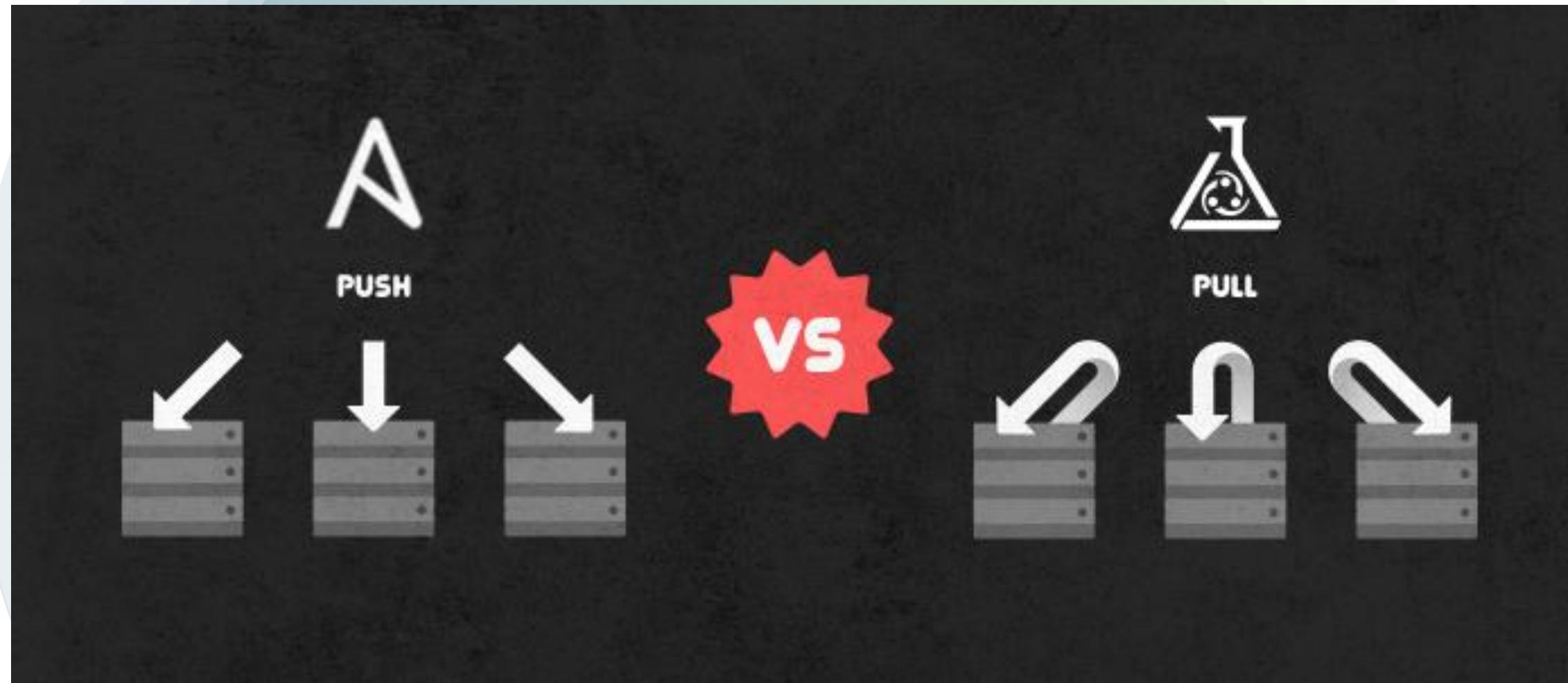
# Terraform Placement



# Declarative vs Imperative (Procedural)



# Pull vs Push Execution Methods



# Terraform vs Ansible



**Terraform**



**builds the house.**



**ANSIBLE**



**makes it a home.**

# Terraform vs Ansible

Feature	Terraform	Ansible
<b>Primary Focus</b>	Infrastructure provisioning and lifecycle management	Configuration management and application deployment
<b>Approach</b>	Declarative (specifies the desired end state)	Procedural (defines steps to reach the desired state)
<b>State Management</b>	Maintains a state file to track infrastructure changes	Does not maintain a state file, relies on idempotent tasks
<b>Language</b>	HCL (HashiCorp Configuration Language)	YAML (human-readable syntax for tasks)
<b>Execution Method</b>	Push-based (initiates changes from a central point)	Push-based (initiates changes from a central point)
<b>Cloud vs. On-Premises</b>	Optimized for cloud infrastructure lifecycle management	Effective for both cloud and on-premises configuration
<b>Modules/Templates</b>	Modules for reusable IaC components	Playbooks and roles for reusable configurations



# Course URLs

Repository Used For	Repository URL
Training GitHub Repository	<a href="https://github.com/salehmiri90/terraform-course.git">https://github.com/salehmiri90/terraform-course.git</a>
Terraform Website	<a href="https://developer.hashicorp.com/terraform">https://developer.hashicorp.com/terraform</a>
Terraform Providers	<a href="https://registry.terraform.io/">https://registry.terraform.io/</a>
Terraform Providers vSphere	<a href="https://github.com/hashicorp/terraform-provider-vsphere">https://github.com/hashicorp/terraform-provider-vsphere</a>

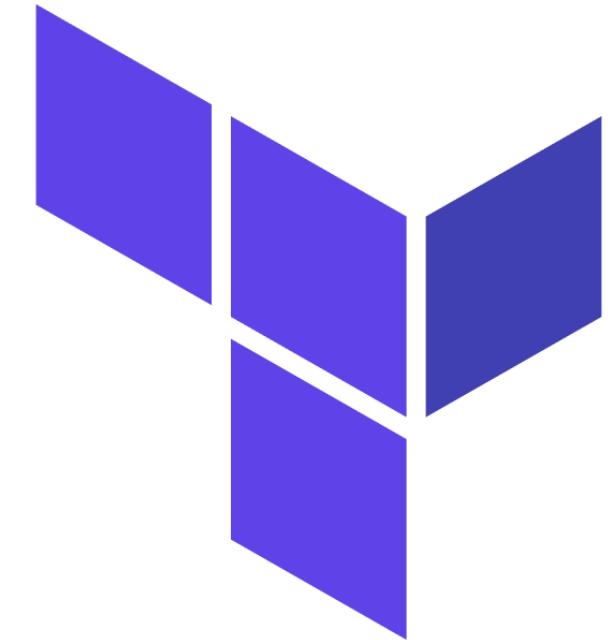
# Terraform Providers



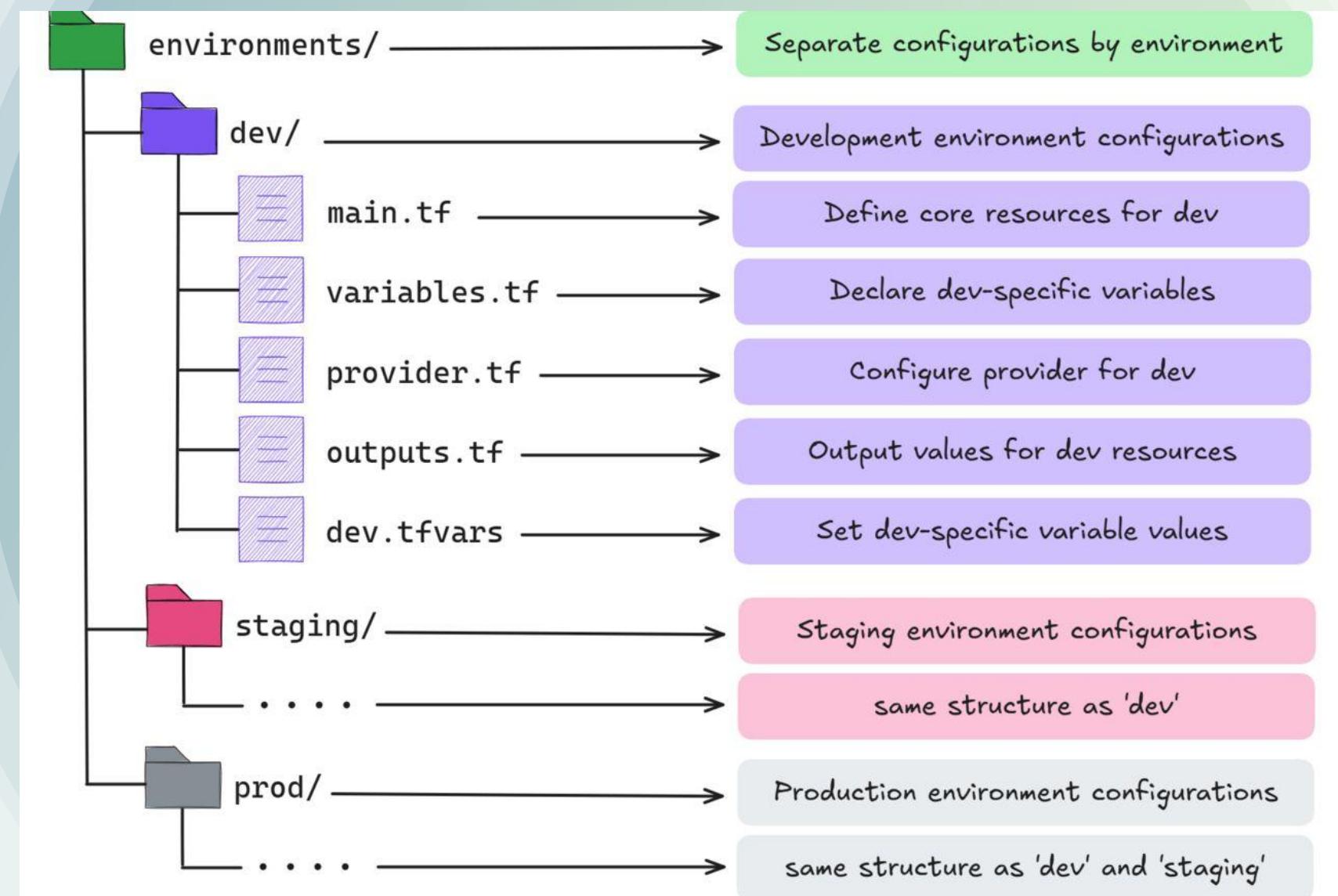
Core focus will be on **mastering Terraform Concepts** with Sample Demo's

# 02 : Terraform Architecture

- **Terraform Directory Structure**
- **Terraform Installation Methods**
- **Terraform Basic Commands**
- **Terraform Workflow**
- **Our first Demo**



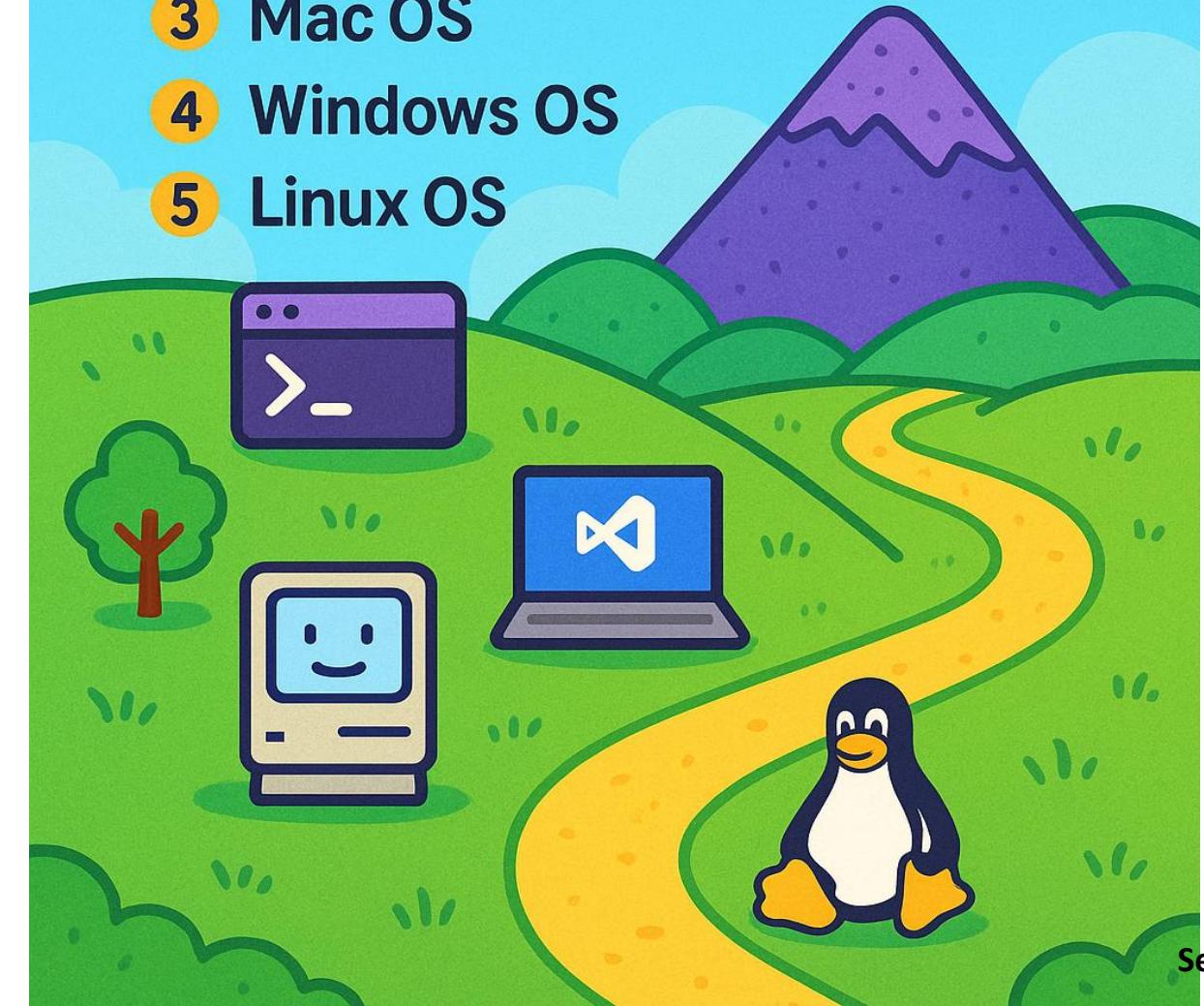
# Terraform Directory Structure



# Terraform Installation Methods

## TERRAFORM INSTALLATION METHODS

- 1 Terraform CLI
- 2 Terraform plugin for VS Code
- 3 Mac OS
- 4 Windows OS
- 5 Linux OS



# Terraform Basic Commands

---

`terraform init`

---

`terraform validate`

---

`terraform plan`

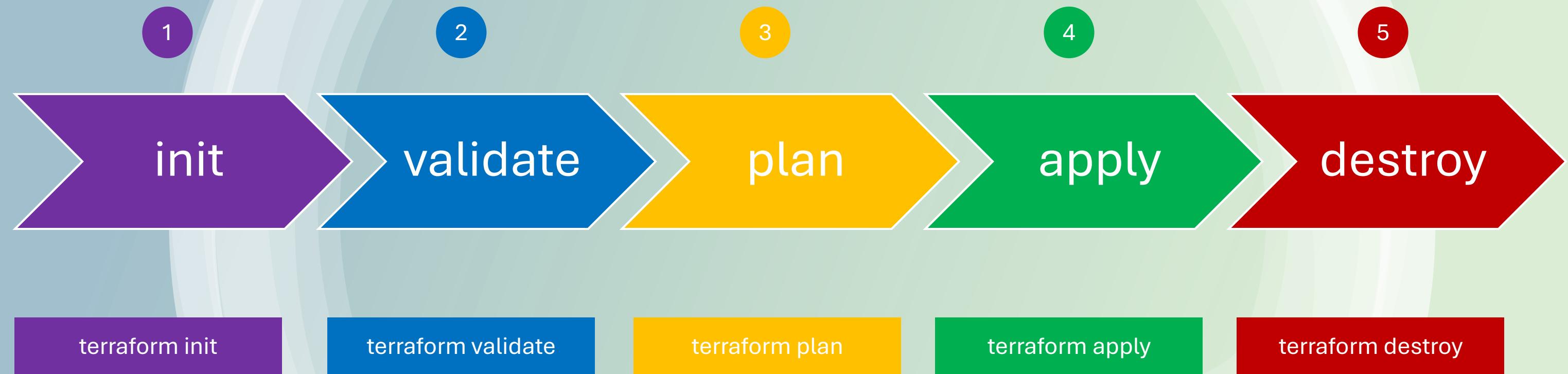
---

`terraform apply`

---

`terraform destroy`

# Terraform Workflow



# Terraform Workflow

1

init

2

validate

3

plan

4

apply

5

destroy

- Initialize a working directory containing terraform config files
- This is the first command that should be run after writing a new Terraform configuration
- Downloads Providers

- Validates the terraform configurations files in that respective directory to ensure they are **syntactically valid** and internally consistent.

- Creates an execution plan
- Terraform performs a refresh and determines what actions are necessary to achieve the **desired state** specified in configuration files

- Used to apply the changes required to reach the desired state of the configuration.
- By default, apply scans the current directory for the configuration and applies the changes appropriately.

- Used to destroy the Terraform-managed infrastructure
- This will ask for confirmation before destroying.



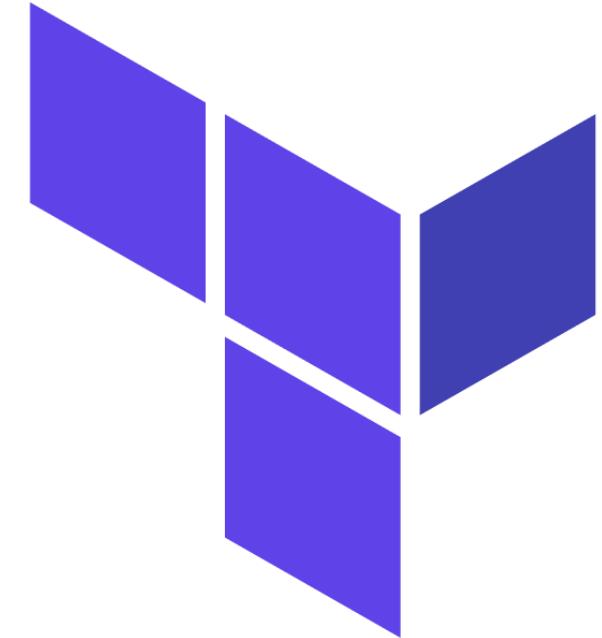
# First Session

[\(click here\)](#)



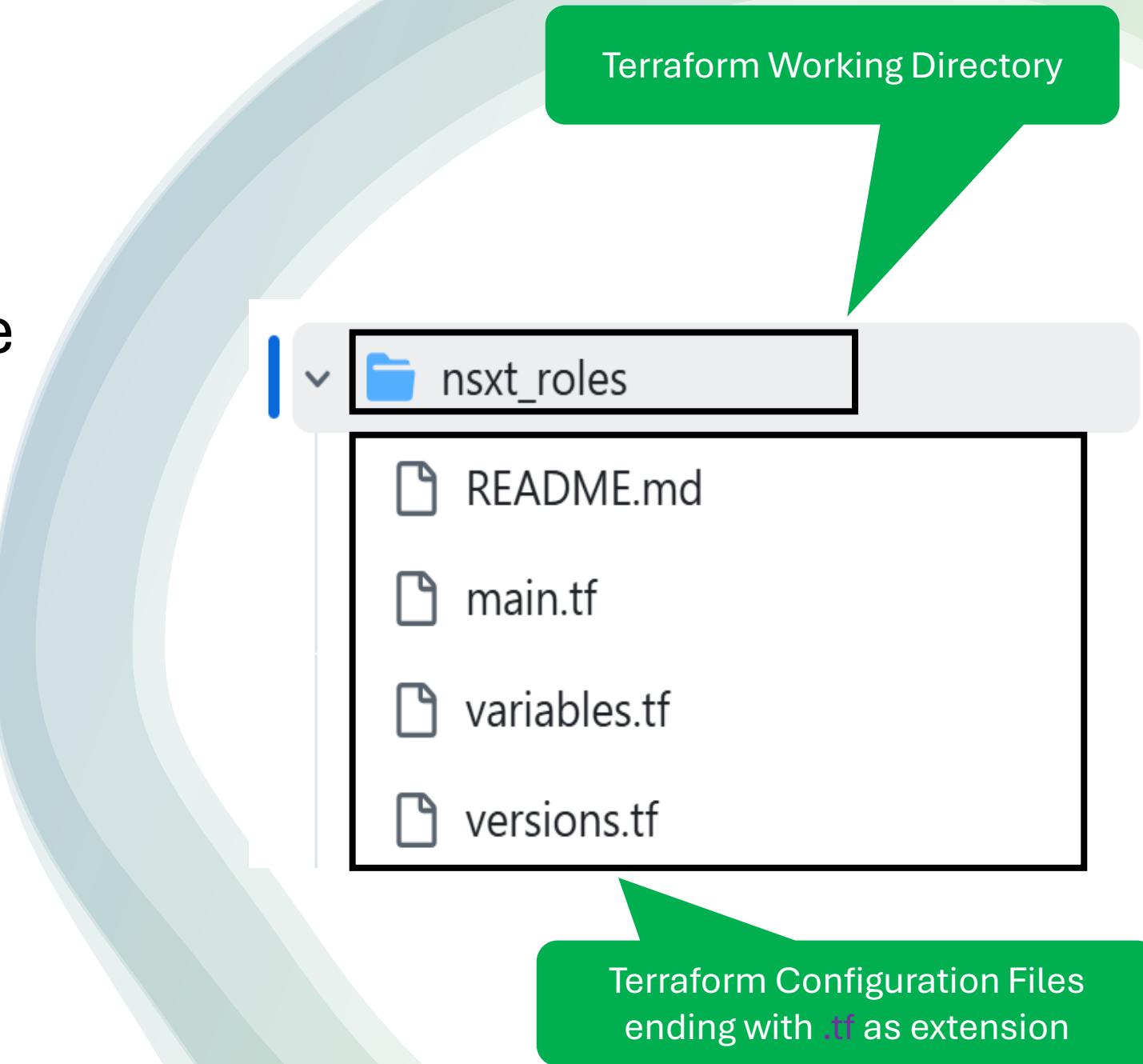
# 02-3 : Terraform Language Syntax

- **Terraform Language Basics – Files**
- **Terraform Language Basics – Configuration Syntax**
- **Terraform Top-Level Blocks**
- **Demo**

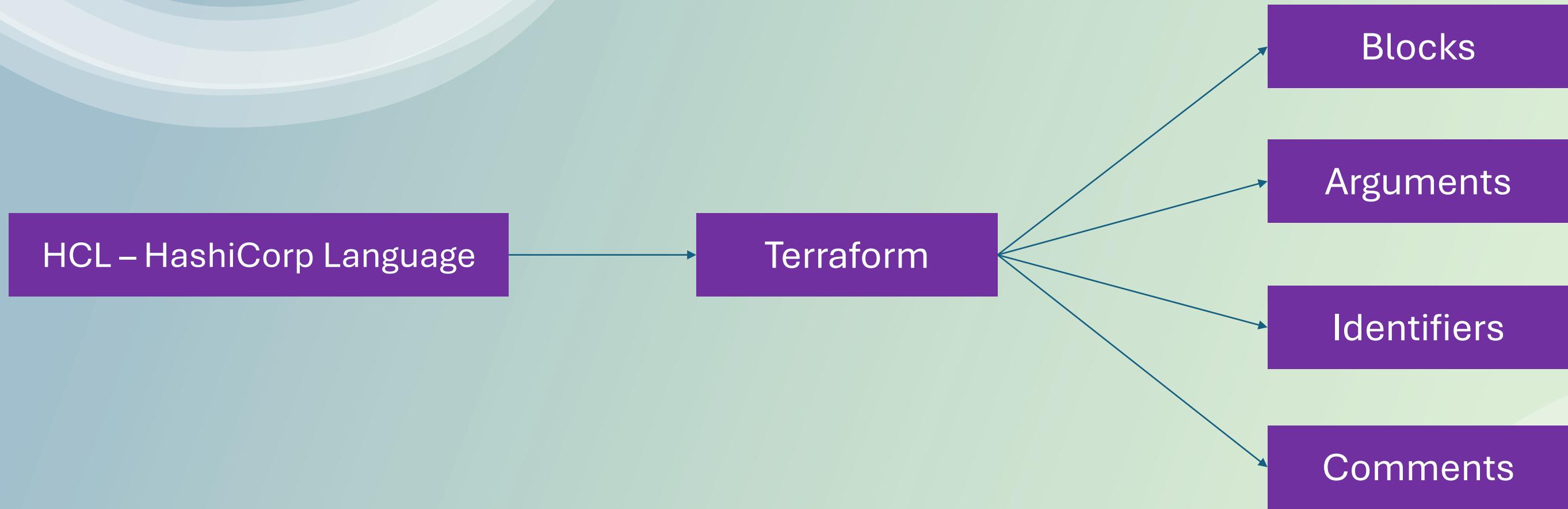


# Terraform Language Basics – Files

- Code in the Terraform language is stored in **plain text files** with the **.tf** file extension.
- There is also a **JSON-based** variant of the language that is named with the **.tf.json** file extension.
- We can call the files containing terraform code as **Terraform Configuration Files** or **Terraform Manifests**



# Terraform Language Basics – Configuration Syntax



# Terraform Language Basics – Configuration Syntax

```
<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>" {  
    # Block body  
    <IDENTIFIER> = <EXPRESSION> # Argument  
}
```

Block Type

Top Level & Block inside Blocks

**Top Level Blocks:** resource, provider

**Block Inside Block:** provisioners, resource specific blocks like tags

```
resource "esxi_guest" "vmsalehmiri01" {  
    guest_name = "vmsalehmiri01"  
    disk_store = "DS_001"  
}
```

Block Labels

Based on Block Type  
block labels will be 1 or 2

**Example:** Resource – 2 labels  
Variables – 1 label

Arguments

# Terraform Language Basics – Configuration Syntax

```
<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>" {  
    # Block body  
    <IDENTIFIER> = <EXPRESSION> # Argument  
}
```

```
resource "esxi_guest" "vmsalehmiri01" {  
    guest_name = "vmsalehmiri01"  
    disk_store = "DS_001"  
}
```

Argument Name  
or  
Identifier

Argument Value  
or  
Expression

# Terraform Language Basics – Configuration Syntax

Single Line Comments with # or //

```
# EC2 Instance Resource
resource "aws_instance" "ec2demo" {
    ami                  = "ami-0885b1f6bd170450c" // Ubuntu 20.04 LTS
    instance_type        = "t2.micro"
    /*
    Multi-line comments
    Line-1
    Line-2
    */
}
```

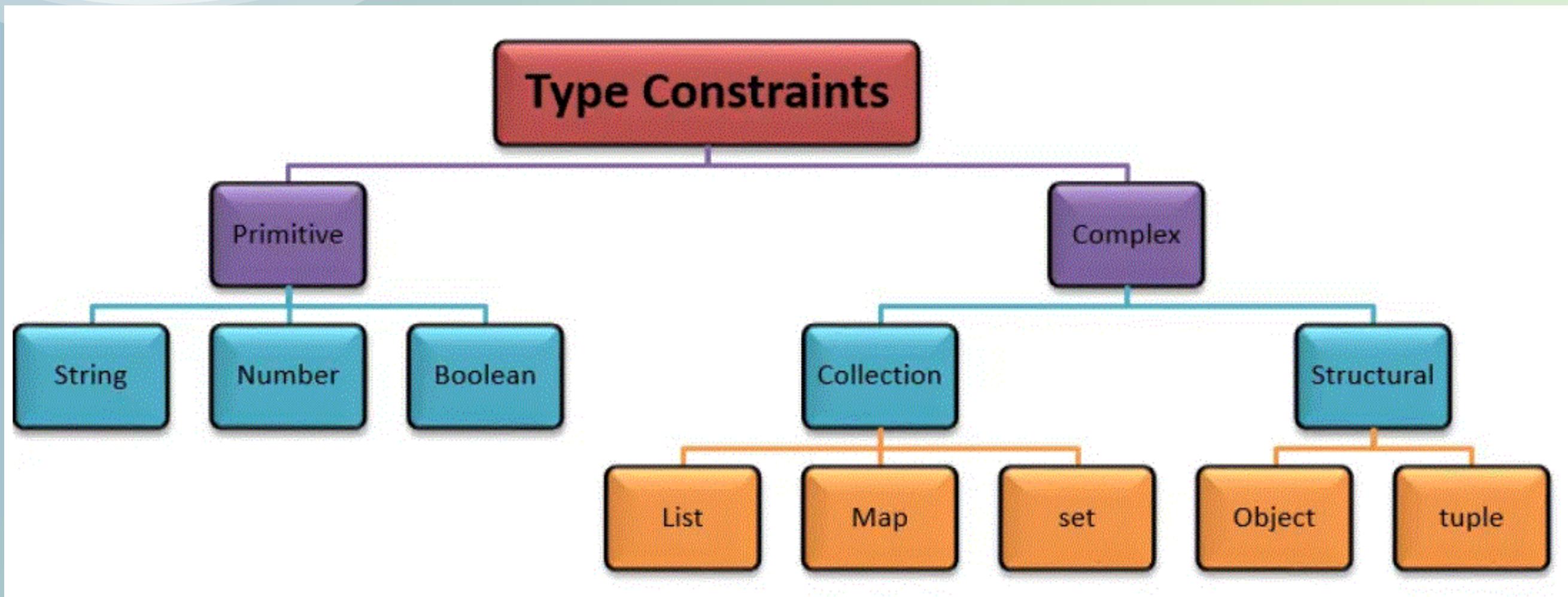
Multi-line comment



WECAMP

Seyed Saleh Miri

# Terraform Language Basics – Configuration Syntax



# List

variables.tf

```
variable "prefix" {  
  default = ["Mr", "Mrs", "Sir"]  
  type = list  0      1      2  
}
```

maint.tf

```
resource "random_pet" "my-pet" {  
  prefix      = var.prefix[0]
```

Index	Value
0	Mr
1	Mrs
2	Sir

# Map

```
variables.tf

variable file-content {
  type      = map
  default   = {
    "statement1" = "We love pets!"
    "statement2" = "We love animals!"
  }
}
```

```
maint.tf

resource local_file my-pet {
  filename  = "/root/pets.txt"
  content   = var.file-content["statement2"]
}
```

Key	Value
statement1	We love pets!
statement2	We love animals!

# Terraform Language Basics – Configuration Syntax

Term	Definition	Examples in Code
Arguments	Input parameters configuring the resource	<code>name</code> , <code>num_cpus</code> , <code>memory</code> , <code>network_interface</code> , <code>clone</code> , <code>connection</code> blocks
Attributes	Output/read-only properties of created resource	<code>self.default_ip_address</code>
Meta-arguments	Special arguments controlling resource behavior	<code>dynamic "disk"</code> with <code>for_each</code> and <code>content</code>



# Terraform Language Basics – Configuration Syntax

Meta-Argument	Purpose	Example
<code>count</code>	Create multiple instances of a resource using a numeric value.	<code>count = 3</code> creates 3 identical VMs.
<code>for_each</code>	Create multiple instances using a map/set of strings (better than <code>count</code> for non-sequential resources).	<code>for_each = toset(["web", "db"])</code> creates resources named <code>web</code> and <code>db</code> .
<code>depends_on</code>	Explicitly define dependencies between resources.	<code>depends_on = [vsphere_folder.parent_folder]</code> waits for folder creation.
<code>lifecycle</code>	Customize how Terraform manages resource lifecycle.	<code>prevent_destroy = true</code> blocks accidental deletion.
<code>provider</code>	Specify non-default provider configurations.	<code>provider = vsphere.eu_west</code> uses a Europe-West vSphere configuration.



Terraform language uses a **limited** number of **top-level block** types, which are **blocks** that can appear **outside** of any other **block** in a TF configuration file.

## Terraform Top-Level Blocks

Most of **Terraform's features** are implemented as **top-level** blocks.

Terraform Block

Providers Block

Resources Block

Fundamental Blocks

Input Variables Block

Output Values Block

Local Values Block

Variable Blocks

Data Sources Block

Modules Block

Calling / Referencing Blocks



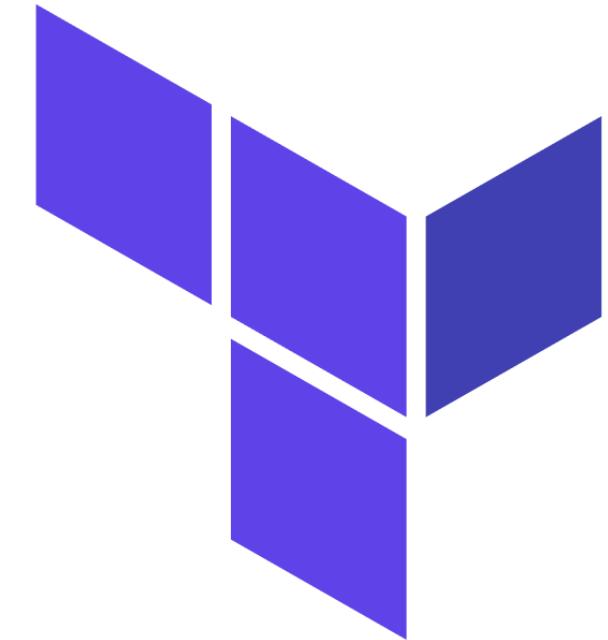
# 2<sup>nd</sup> Session

[\(click here\)](#)



# 03 : Terraform Fundamental Blocks

- **Terraform Basic Blocks**
- **Terraform Blocks**
- **Demo**



# Terraform Basic Blocks

## Terraform Block

Special block used to configure some behaviors

Specifying a required Terraform Version

Specifying Provider Requirements

Configuring a Terraform Backend (Terraform State)

## Provider Block

HEART of Terraform

Terraform relies on providers to interact with Remote Systems

Declare providers for Terraform to install providers & use them

Provider configurations belong to Root Module

## Resource Block

Each Resource Block describes one or more Infrastructure Objects

Resource Syntax: How to declare Resources?

Resource Behavior: How Terraform handles resource declarations?

Provisioners: We can configure Resource post-creation actions



# Terraform Block

- This block can be called in 3 ways. All means the same.
  - Terraform Block
  - Terraform Settings Block
  - Terraform Configuration Block
- Each terraform block can contain a number of settings related to Terraform's behavior.
- Within a terraform block, **only constant values can be used**; arguments **may not refer** to named objects such as resources, input variables, etc, and **may not use any** of the Terraform language built-in functions.

# Terraform Block from 0.13 onwards

Terraform 0.12 and earlier:

```
# Configure the AWS Provider
provider "aws" {
  version = "~> 3.0"
  region  = "us-east-1"
}

# Create a VPC
resource "aws_vpc" "example" {
  cidr_block = "10.0.0.0/16"
}
```

Terraform 0.13 and later:

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
}

# Configure the AWS Provider
provider "aws" {
  region = "us-east-1"
}

# Create a VPC
resource "aws_vpc" "example" {
  cidr_block = "10.0.0.0/16"
}
```

# Terraform Block

Terraform Block

Required Terraform Version

Provider Requirements

Terraform Backend

Experimental Language  
Features

Passing Metadata to  
Providers

```
terraform {  
    # Required Terraform Version  
    required_version = "~> 0.14.3"  
  
    # Required Providers and their Versions  
    required_providers {  
        aws = {  
            source  = "hashicorp/aws"  
            version = "~> 3.21" # Optional but recommended  
        }  
    }  
  
    # Remote Backend for storing Terraform State in S3 bucket  
    backend "s3" {  
        bucket = "mybucket"  
        key    = "path/to/my/key"  
        region = "us-east-1"  
    }  
  
    # Experimental Features (Not required)  
    experiments = [ example ]  
  
    # Passing Metadata to Providers (Super Advanced – Terraform 0.14+)  
    provider_meta "my-provider" {  
        hello = "world"  
    }  
}
```



WECAMP

Seyed Saleh Miri

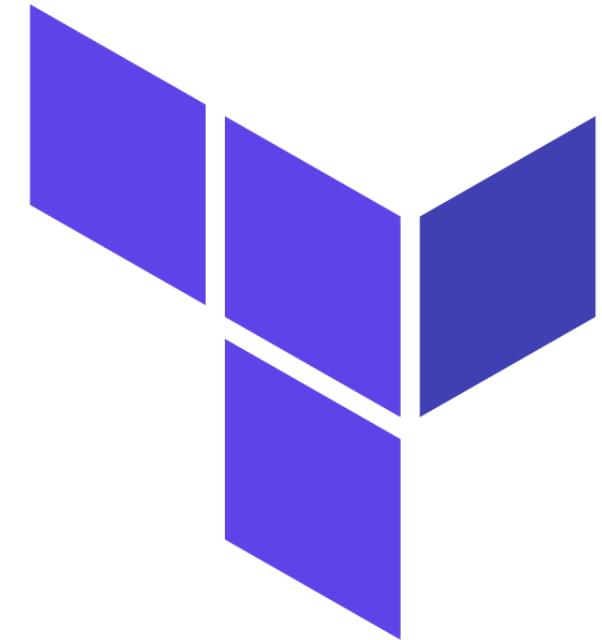
# 3<sup>rd</sup> Session

[\(click here\)](#)



## 03-2: Terraform Providers

- **Terraform Providers**
- **Dependency Lock File**
- **Required Providers**
- **Terraform Registry**
- **Terraform Registry**
- **Demo**



# Terraform Providers

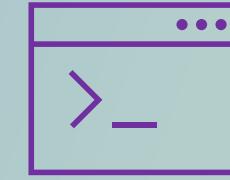
Terraform Admin



Local Desktop



Terraform CLI



Terraform vSphere Provider



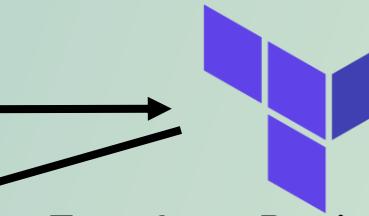
2 terraform validate

3 terraform plan

1

terraform init

Download Provider



Terraform Registry

4 terraform apply

5 terraform destroy

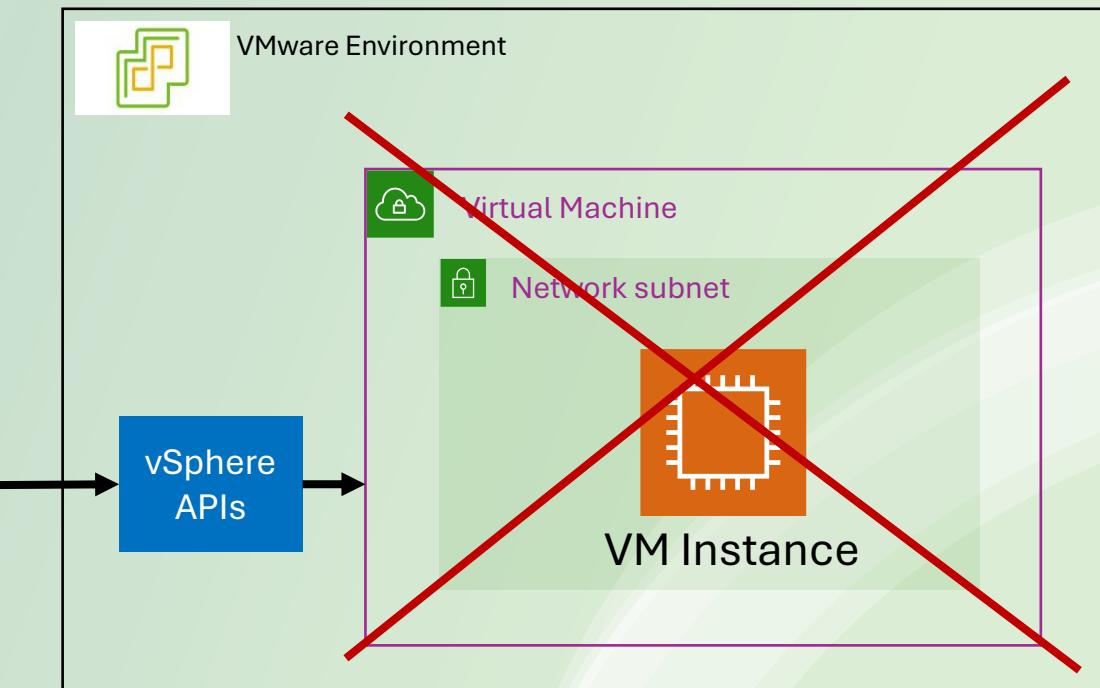
Providers are **HEART** of Terraform

Every Resource Type (example: VM, EC2), is implemented by a Provider

Without Providers Terraform **cannot** manage any infrastructure.

Providers are distributed separately from Terraform and each provider has its own **release cycles** and **Version Numbers**

Terraform **Registry** is publicly available which contains many Terraform Providers for most **major** Infra Platforms



## Terraform Providers

## Provider Requirements

```
# Terraform Settings Block
terraform {
  required_version = "~> 1.8.0"
  required_providers {
    vsphere = {
      source  = "hashicorp/vsphere"
      version = "2.8.1"
    }
  }
}
```

## Provider Configuration

```
# Provider Block
provider "vsphere" {
  user          = "administrator@vsphere.local"
  password      = "P@ssw0rd"
  vsphere_server = "1.2.3.4"
  allow_unverified_ssl = true
}
```

## Dependency Lock File

```
└── terraform-manifests
    ├── .terraform
    └── .terraform.lock.hcl
```

```
└── ec2-instance.tf
```

```
└── terraform.tfstate
```

```
└── terraform.tfstate.backup
```

```
# This file is maintained automatically by "terraform init".
# Manual edits may be lost in future updates.

provider "registry.terraform.io/hashicorp/aws" {
  version = "3.22.0"
  hashes = [
    "h1:f/TzZv1Zb78ZaiyJkQ0MGIViZwbYrLuQk3kojPM91c=",
    "zh:4a9a66caf1964cd3b61fb3ebb0da417195a5529cb8e496f266b0778335d11c8",
    "zh:514f2f006ae68db715d86781673faf9483292deab235c7402ff306e0e92ea11a",
    "zh:5277b61109fdbb9011728f6650ef01a639a0590effe34ed7de7ba10d0c31803",
    "zh:67784dc8c8375ab37103eea1258c3334ee92be6de033c2b37e3a2a65d0005142",
    "zh:76d4c8be2ca4a3294fb51fb58de1fe03361d3bc403820270cc8e71a04c5fa806",
    "zh:f89f0b1cfdf6e8fb1a9d0382ecaa5056a3a84c94e313fb9e92c89de271cde",
    "zh:d0ac346519d0df124df89be2d803eb53f373434890f6ee3fb37112802f9eac59",
    "zh:db256feedada82cbfb1bd6d9ad02048f23120ab50e6146a541cb11a108cc1",
    "zh:db2fe0d2e77c02e9a74e1ed694aa352295a50283f9a1cf896e5be252af14e9f4",
    "zh:eda61e889b579bd90046939a5b40cf5dc9031fb5a819fc3e4667a78bd432bdb2"
  ]
}
```



# Provider Requirements

```
# Terraform Settings Block
terraform {
  required_version = "~> 1.5.0"
  required_providers {
    vsphere = {
      source  = "hashicorp/vsphere"
      version = "2.8.1"
    }
  }
}
```

## Local Names

Local Names are Module specific and should be **unique per-module**

Terraform configurations always refer to **local name** of provider **outside** required\_provider block

Users of a provider can choose **any local name** for it (myvsphere, vsphere1, vsphere2, provider1)

Recommended way of choosing local name is to use preferred local name of that provider  
(For vSphere Provider: hashicorp/vsphere, **preferred local name** is vsphere)

```
# Provider Block
provider "vsphere" {
  user        = "administrator@vsphere.local"
  password    = "P@ssw0rd"
  vsphere_server = "1.2.3.4"
  allow_unverified_ssl = true
}
```

## Source

It is the **primary location** where we can download the Terraform Provider

Source addresses consist of three parts delimited by **slashes (/)**

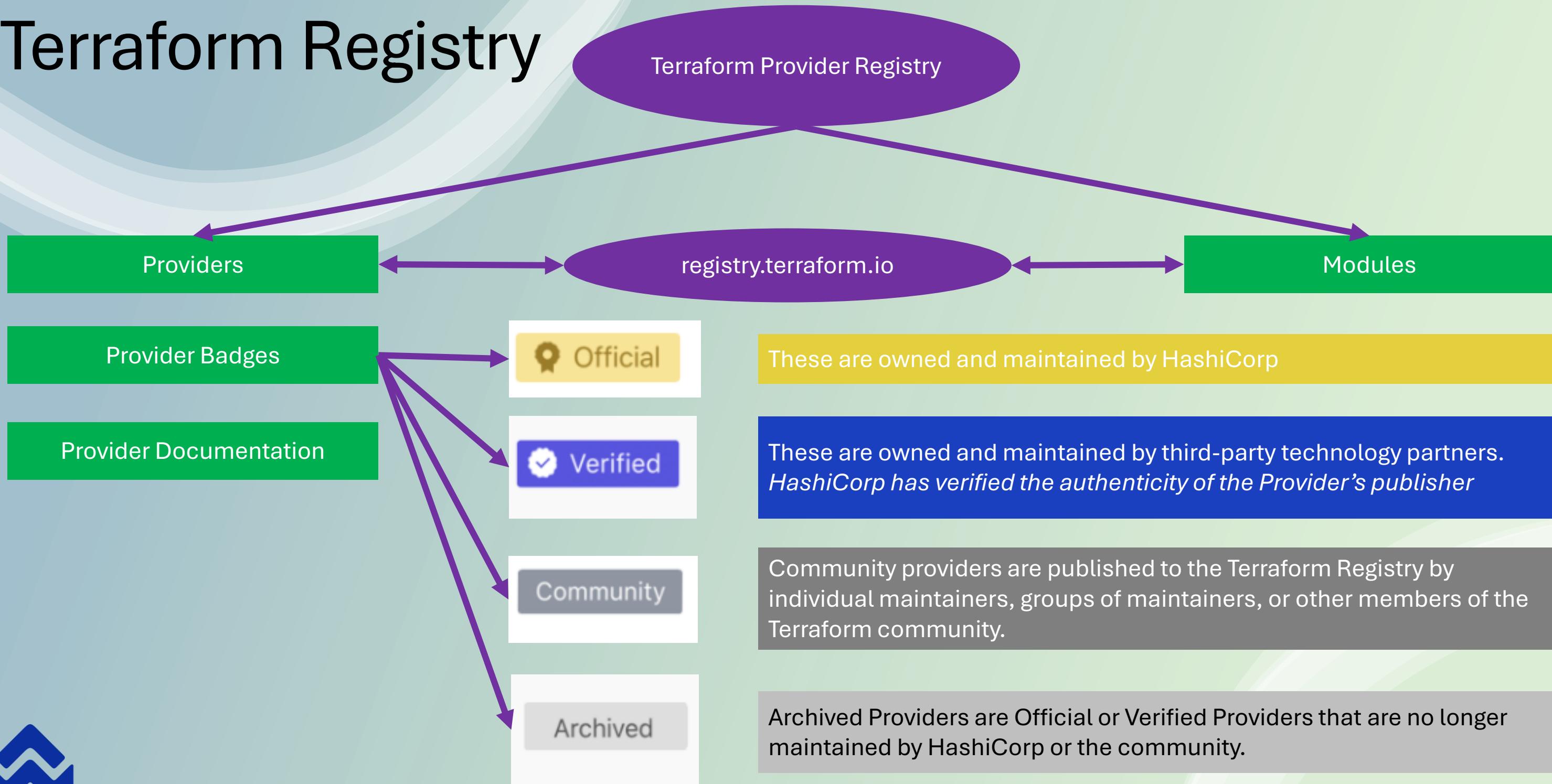
[<HOSTNAME>/]<NAMESPACE>/<TYPE>

registry.terraform.io/hashicorp/vsphere

Registry Name is **optional** as default is going to be Terraform Public Registry



# Terraform Registry



# Dependency Lock File

Terraform

Providers

This feature added from Terraform v0.14 & later

Version Constraints within the configuration itself determine which versions of dependencies are *potentially* compatible

Dependency Lock File: After selecting a specific version of each dependency using Version Constraints Terraform remembers the decisions it made in a dependency lock file so that it can (by default) make the same decisions again in future.

Location of Lock File: Current Working Directory

Very Important: Lock File currently tracks only Provider Dependencies.

Checksum Verification: Terraform will also verify that each package it installs matches at least one of the checksums it previously recorded in the lock file, if any, returning an error if none of the checksums match



WECAMP

Seyed Saleh Miri

# Multiple Providers

We can define **multiple configurations** for the **same provider**, and select which one to use on a **per-resource** or **per-module** basis.

The primary reason for this is to **support multiple regions** for a **cloud platform**

We can **use** the alternate provider in a resource, data or module by referencing it as **<PROVIDER NAME>.<ALIAS>**

```
# Provider-1 for vcenter1 (Default Provider)
provider "vsphere" {
  user           = "administrator@vsphere.local"
  password       = "P@ssw0rd"
  vsphere_server = "1.2.3.4"
  allow_unverified_ssl = true
}

# Provider-2 for vcenter2
provider "vsphere" {
  user           = "sre@vsphere.local"
  password       = "1404AdvancePass"
  vsphere_server = "11.12.13.14"
  allow_unverified_ssl = true
  alias          = "cluster2"
}
```

```
# Resource Block to Create VM in vcenter2
resource "vsphere_virtual_machine" "resource2_vm" {
  name      = "resource2_vm"
  #<PROVIDER NAME>.<ALIAS>
  provider  = vsphere.cluster2
}
```



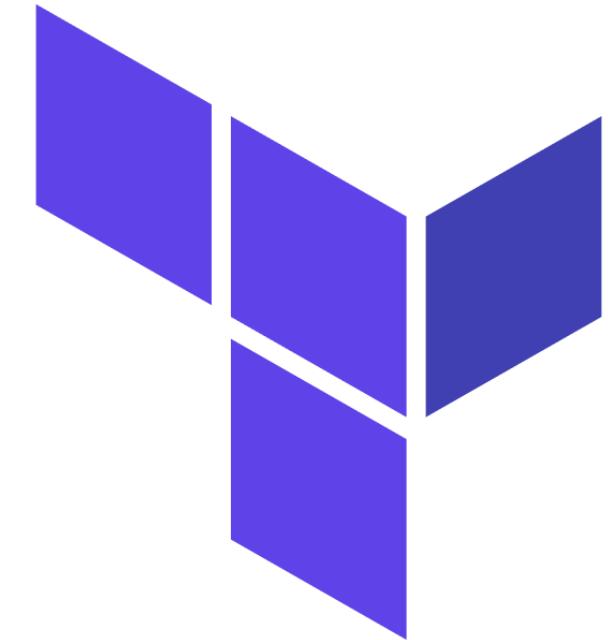
# 4<sup>th</sup> Session

[\(click here\)](#)



# 04 : Terraform Resources

- **Terraform Language Basics**
- **Terraform Resource Syntax**
- **Terraform Resource Behavior**
- **Terraform State**
- **Meta-Argument count**
- **Meta-Argument depends\_on**
- **Meta-Argument for\_each**
- **Meta-Argument lifecycle**
- **Demo**



# Terraform Language Basics – Configuration Syntax

```
<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>" {  
    # Block body  
    <IDENTIFIER> = <EXPRESSION> # Argument  
}
```

Block Type

resource

"esxi\_guest" "vmsalehmiri01"

Block Labels

guest\_name = "vmsalehmiri01"

disk\_store = "DS\_001"

Arguments

}

Argument Name

Argument Value



# Resource Syntax

**Resource Type:** It determines the kind of infrastructure object it manages and what arguments and other attributes the resource supports.

**Resource Local Name:** It is used to refer to this resource from elsewhere in the same Terraform module, but has no significance outside that module's scope.  
The resource type and name together serve as an identifier for a given resource and so must be unique within a module

**Meta-Arguments:** Can be used with any resource to change the behavior of resources

**Resource Arguments:** Will be specific to resource type.  
Argument Values can make use of Expressions or other Terraform Dynamic Language Features

```
# Provider Block
provider "vsphere" {
  user           = "administrator@vsphere.local"
  password       = "P@ssw0rd"
  vsphere_server = "1.2.3.4"
  allow_unverified_ssl = true
  alias          = "cluster2"
}

# Resource Block
resource "vsphere_virtual_machine" "saleh_vm" {
  provider = vsphere.cluster2

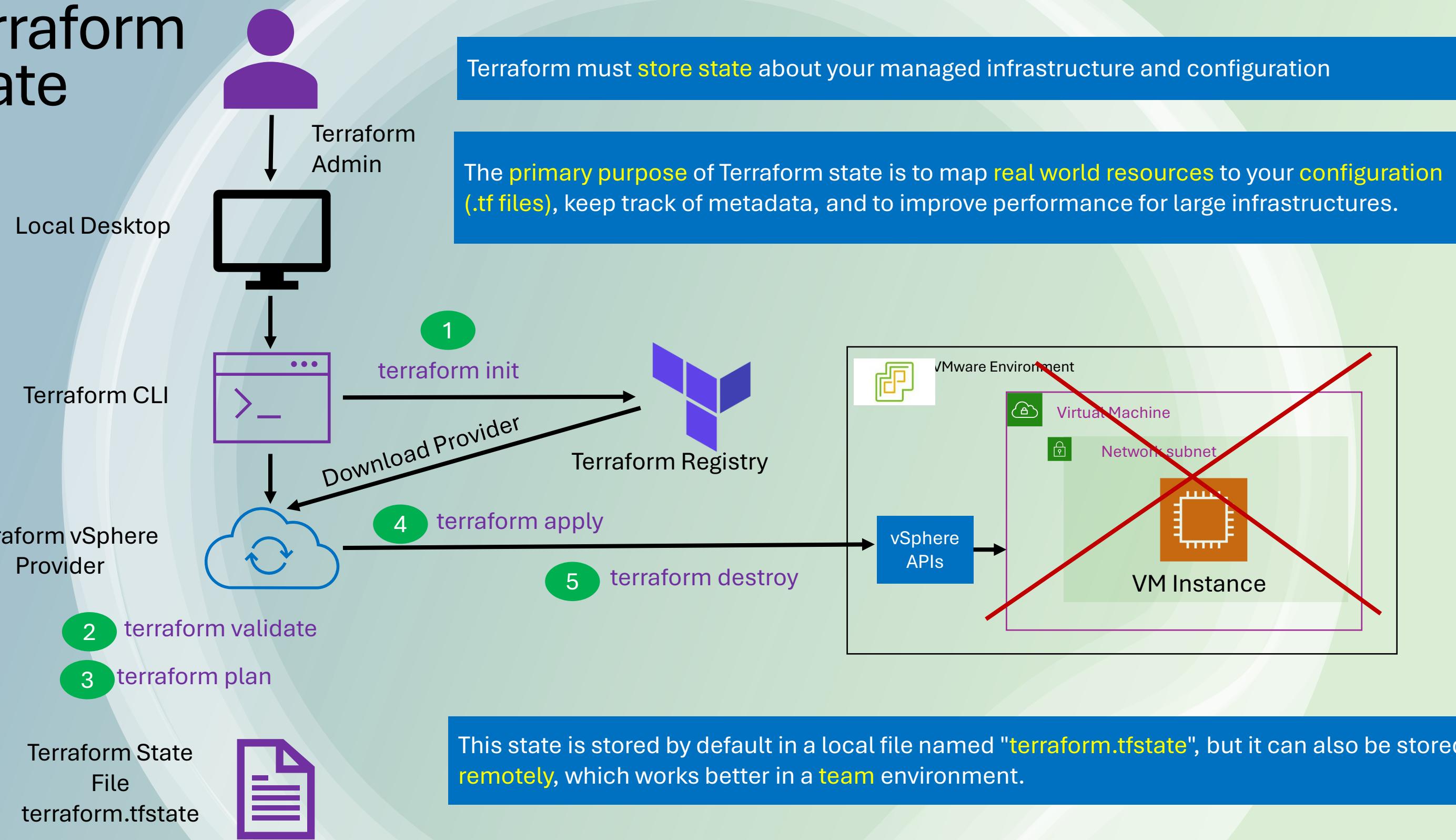
  name           = "saleh-vm"
  datastore_id   = data.vsphere_datastore.ds.id
  num_cpus       = 4
  memory         = 4096
  #...
}
```

# Resource Behavior



## Terraform State

# Terraform State



# Desired & Current Terraform States

Real World Resource – VM Instance

saleh-vm

VMware Tools is not installed on this virtual machine.

Virtual Machine Details

- Power Status: Powered On
- Guest OS: Other (32-bit)
- VMware Tools: Not running, not installed

VM Hardware

CPU	4 CPU(s), 0 MHz used
Memory	4 GB, 0 GB memory active
Hard disk 1	10 GB   Thin Provision vsanDatastore
Network adapter 1	VM Network (connected)   00:50:56:b3:64:1c

LAUNCH REMOTE CONSOLE

LAUNCH WEB CONSOLE

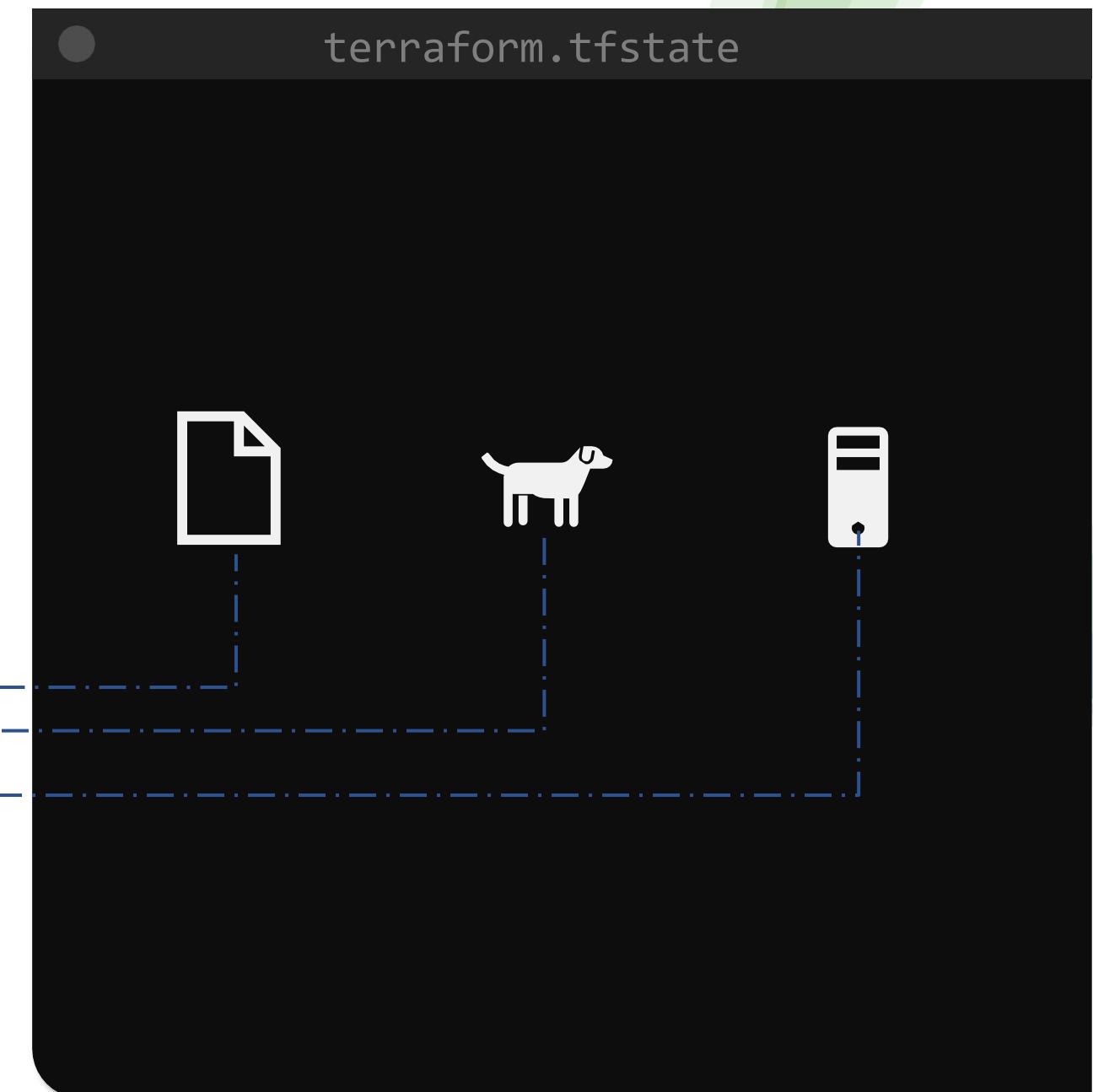
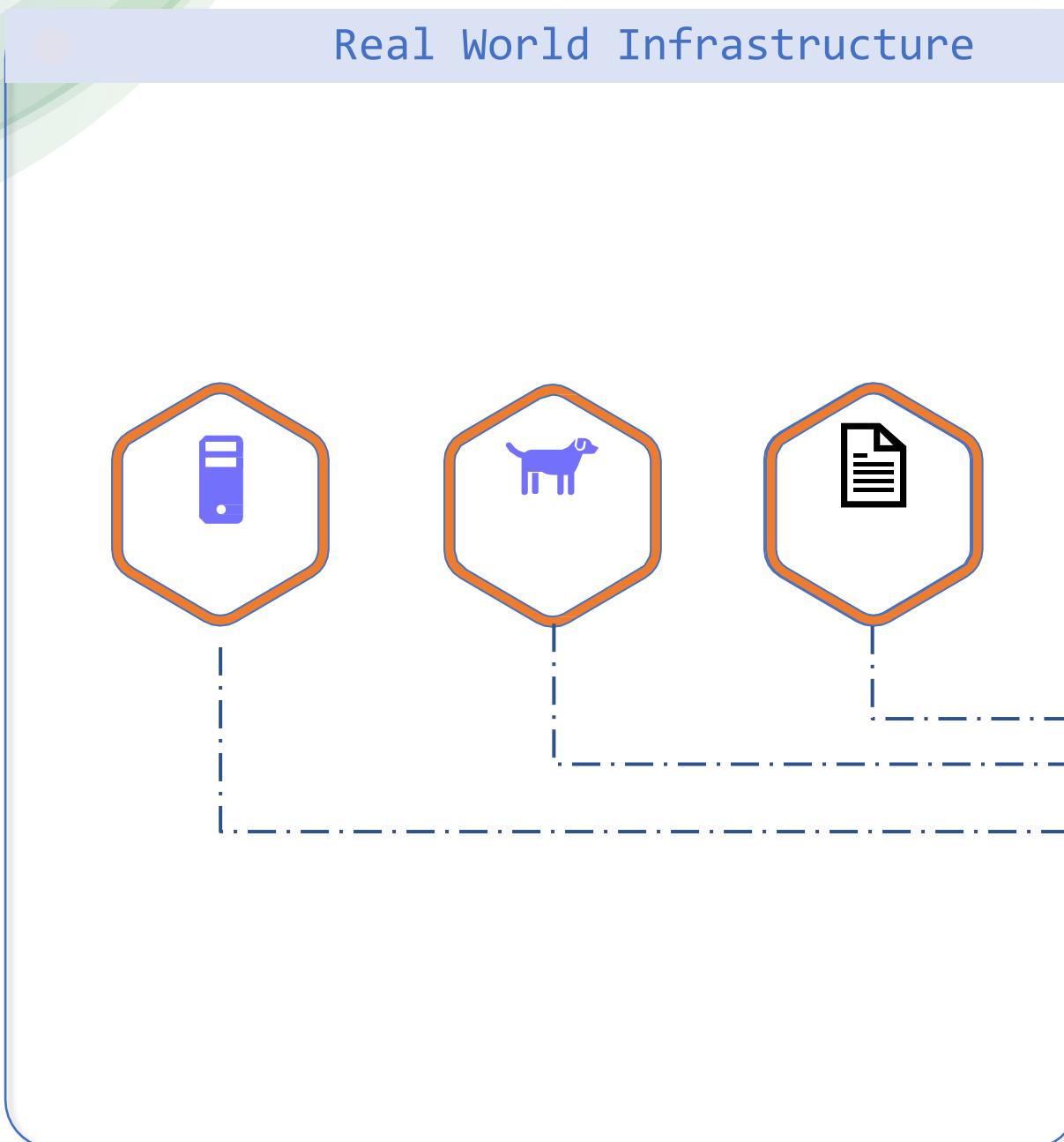
Current State

Terraform Configuration Files



```
terraform-manifests-1
  c1-versions.tf
  c2-saleh-vm.tf
```

Desired State



```
variables.tf

variable "filename" {
  default = "/root/pets.txt"
}
variable "content" {
  default = "We love pets!"
}
```

```
>_
$ terraform plan

Refreshing Terraform state in-memory
prior to plan...
The refreshed state will be used to
calculate this plan, but will not be
persisted to local or remote state
storage.

local_file.pet: Refreshing state...
[id=7e4db4fbfdbb108bdd04692602bae3e9bd1e
1b68]
.
.
.
[Output Truncated]
```



```
>_
[terraform-local-file]$ cat  terraform.tfstate
{
  "version": 4,
  "terraform_version": "0.13.0",
  "serial": 1,
  "lineage": "e35dde72-a943-de50-3c8b-1df8986e5a31",
  "outputs": {},
  "resources": [
    {
      "mode": "managed",
      "type": "local_file",
      "name": "pet",
      "provider": "provider[\\"registry.terraform.io/hashicorp/local\\"]",
      "instances": [
        {
          "schema_version": 0,
          "attributes": {
            "content": "I love pets!",
            "content_base64": null,
            "directory_permission": "0777",
            "file_permission": "0777",
            "filename": "/root/pets.txt",
            "id": "7e4db4fbfdbb108bdd04692602bae3e9bd1e1b68",
            "sensitive_content": null
          },
          "private": "bnVsbA=="
        }
      ]
    }
  ]
}
```

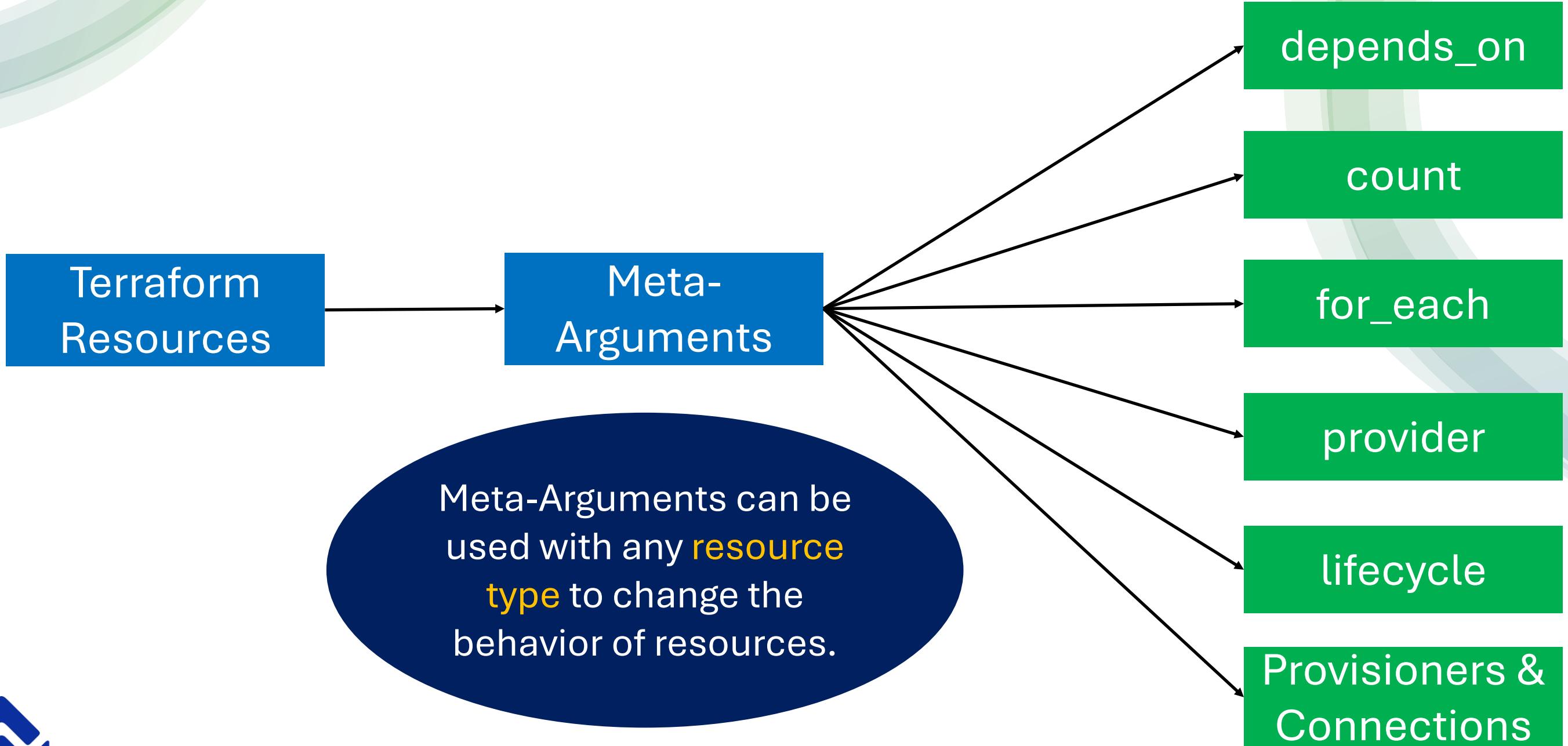


# 5<sup>th</sup> Session

[\(click here\)](#)



# Resource Meta-Arguments



# Resource Meta-Arguments

`depends_on`

Explicitly defines dependencies between resources to control the creation order.

`count`

Creates multiple instances of a resource based on the given number.

`for_each`

Creates multiple instances of a resource using a map or set of strings as input.

`provider`

Specifies which provider configuration to use for a resource.

`lifecycle`

Configures how Terraform handles resource creation, update, and deletion.

`Provisioners`

Executes scripts or commands on a local or remote machine during resource creation or destruction.

`Connections`

Defines how Terraform connects to the resource for provisioning (e.g., SSH or WinRM).



# Resource Meta-Arguments – depends\_on

Use the `depends_on` meta-argument to handle `hidden` resource or module dependencies that Terraform can't automatically infer.

Explicitly specifying a dependency is only necessary when a resource or module relies on some other resource's behavior but `doesn't access` any of that resource's data in its arguments.

## Resource Meta-Argument `depends_on`

The `depends_on` meta-argument, if present, must be a list of references to `other resources` or `child modules` in the same calling module.

Arbitrary expressions `are not allowed` in the `depends_on` argument value, because its value must be known before Terraform knows resource relationships and thus before it can safely evaluate expressions.

This argument is available in `module blocks` and in all `resource blocks`, regardless of resource type.

The `depends_on` argument should be used only as a `last resort`. Add comments for future reference about why we added this.



# Use case: What are we going implement?

# Scenario:  
create a virtual machine with Terraform in vSphere only if a custom folder (named saleh-folder) already exists.

# Resource-1: Create a resource for vSphere Folder

Use `depends_on` to set an explicit dependency on mentioned Folder

# Resource-2:  
Create VM  
`depends_on`

# Resource-2: Create VM Instance in the selected Folder

```
resource "vsphere_virtual_machine" "saleh_vm" {  
    name          = "saleh-vm"  
    folder        = vsphere_folder.saleh_folder.path  
    depends_on   = [vsphere_folder.saleh_folder]  
    resource_pool_id = data.vsphere_compute_cluster.  
    datastore_id    = data.vsphere_datastore.ds.id  
  
    num_cpus      = 4  
    memory        = 4096
```

# Resource Meta-Arguments – count

If a resource or module block includes a **count** argument whose value is a whole number, Terraform will create that many instances.

**count.index:** The distinct index number (starting with 0) corresponding to this instance.

Each instance has a distinct infrastructure object associated with it, and each is separately created, updated, or destroyed when the configuration is applied.

Resource  
Meta-Argument  
count

When count is set, Terraform distinguishes between the block itself and the multiple resource or module instances associated with it. Instances are identified by an index number, starting with 0.  
**vsphere\_virtual\_machine.saleh\_vm[0]**

The count meta-argument accepts numeric expressions. The count value must be known *before* Terraform performs any remote resource actions.

Module support for count was added in Terraform 0.13, and previous versions can only use it with resources.

A given resource or module block **cannot** use both count and for\_each

# Use case: What are we going implement?

# Resource-1: Use Meta-Argument Count to create multiple VM Instances using single Resource

```
# Resource Block
resource "vsphere_virtual_machine" "saleh_vm" {

    count      = 5
    name       = "saleh-vm-${count.index}"
    num_cpus   = 4
    memory     = 4096
```

count

count.index

vsphere\_virtual\_machine.saleh\_vm[0]

vsphere\_virtual\_machine.saleh\_vm[1]

vsphere\_virtual\_machine.saleh\_vm[2]

vsphere\_virtual\_machine.saleh\_vm[3]

vsphere\_virtual\_machine.saleh\_vm[4]



# Resource Meta-Arguments – `for_each`

If a resource or module block includes a `for_each` argument whose value is a map or a set of strings, Terraform will create one instance for each member of that map or set.

A given resource or module block **cannot** use both `count` and `for_each`

For set of Strings, each.key = each.value  
`for_each = toset( ["Saleh", "Miri"] )`  
each.key = Saleh  
each.key = Miri

## Resource Meta-Argument `for_each`

Each instance has a distinct infrastructure object associated with it, and each is separately created, updated, or destroyed when the configuration is applied.

For Maps, we use `each.key` & `each.value`  
`for_each = {`  
  `dev = "my-vm"`  
`}`  
each.key = dev  
each.value = my-vm

In blocks where `for_each` is set, an additional `each` object is available in expressions, so you can modify the configuration of each instance.  
`each.key` — The map key (or set member) corresponding to this instance.  
`each.value` — The map value corresponding to this instance. (If a set was provided, this is the same as `each.key`.)

Module support for `for_each` was added in Terraform 0.13, and previous versions can only use it with resources.

# for\_each vs count



WECAMP

Seyed Saleh Miri

# Usecase-1: for\_each Maps

```
# Resource-1: Use Meta-Argument for_each with Maps to  
create multiple VMs using single Resource
```

Define **for\_each** with Map with Key Value pairs

Use **each.key** and **each.value** for the  
VM name

Use **each.key** and **each.value** for VM disk  
names

- ⇨ saleh-vm-a
- ⇨ saleh-vm-b
- ⇨ saleh-vm-c
- ⇨ saleh-vm-d
- ⇨ saleh-vm-e

```
# Resource Block  
resource "vsphere_virtual_machine" "saleh_vm" {  
    // added in this section  
  
    for_each = {  
        "vm-a" = 5  
        "vm-b" = 10  
        "vm-c" = 15  
        "vm-d" = 20  
        "vm-e" = 25  
    }  
  
    name      = "saleh-${each.key}"  
    disk {  
        label          = "saleh-${each.key}-disk0"  
        size           = each.value  
        eagerly_scrub = false  
        thin_provisioned = true  
    }  
    //
```



# Usecase-2: for\_each Set of Strings (toset)

# Resource-1: Use Meta-Argument `for_each` with Set of Strings to create multiple VMs using single Resource

```
# Variable Block
variable "vm_names" {
  default = ["tosetvm-01", "tosetvm-02", "tosetvm-03"]
}

# Resource Block
resource "vsphere_virtual_machine" "saleh_vm" {

  for_each = toset(var.vm_names)
  name    = "saleh-${each.key}"
  //
```

lifecycle is a **nested block** that can appear within a resource block

## Resource Meta-Argument lifecycle

The lifecycle block and its contents are **meta-arguments**, available for all resource blocks regardless of **type**.

create\_before\_destroy

```
data "vsphere_datastore" "ds" {  
    // change datastore name from dat  
    name      = "vsanDatastore"  
    # name      = "datastore2"  
    datacenter_id = data.vsphere_data  
}  
  
// added in this section  
lifecycle {  
    create_before_destroy = true  
}  
//
```

prevent\_destroy

ignore\_changes

```
# Resource Block  
resource "vsphere_virtual_machine" "saleh_vm" {  
    name      = "saleh-lifecycle2"  
    num_cpus  = 1  
    memory   = 1024  
    lifecycle {  
        prevent_destroy = true # Default is false  
    }  
}
```

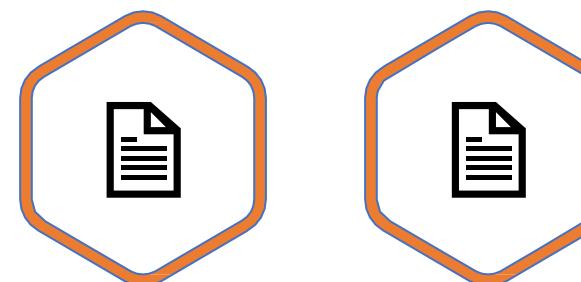
```
# Resource Block  
resource "vsphere_virtual_machine" "saleh_vm" {  
    name      = "saleh-lifecycle3"  
    #name     = "saleh-lifecycle3-new"  
    lifecycle {  
        ignore_changes = [name]  
    }  
}
```



# create\_before\_destroy

main.tf

```
resource "local_file" "pet" {  
    filename = "v1-pets.txt"  
    content = "We love pets!"  
    file_permission = "0700"  
  
    lifecycle {  
        create_before_destroy = true  
    }  
  
}
```



>\_

```
$ terraform apply  
  
# local_file.pet must be replaced  
-/+ resource "local_file" "pet" {  
    content          = "We love pets!"  
    directory_permission = "0777"  
    ~ file_permission = "0777" -> "0755" # forces repl  
    filename         = "v1-pet.txt"  
    ~ id              =  
"5f8fb950ac60f7f23ef968097cda0a1fd3c11bdf" -> (known after ap  
    )  
  
Plan: 1 to add, 0 to change, 1 to destroy.  
  
...  
  
local_file.pet: Creating...  
local_file.pet: Creation complete after 0s  
[id=5f8fb950ac60f7f23ef968097cda0a1fd3c11bdf]  
  
local_file.pet: Destroying...  
[id=5f8fb950ac60f7f23ef968097cda0a1fd3c11bdf]  
local_file.pet: Destruction complete after 0s  
  
Apply complete! Resources: 1 added, 0 changed, 1 destroyed.
```

# prevent\_destroy

main.tf

```
resource "local_file" "pet" {  
    filename = "v2-pets.txt"  
    content = "We love pets!"  
    file_permission = "0700"  
  
    lifecycle {  
        prevent_destroy = true  
    }  
}
```

>\_

```
$ terraform apply  
local_file.my-pet: Refreshing state...  
[id=cba595b7d9f94ba1107a46f3f731912d95fb3d2c]  
  
Error: Instance cannot be destroyed  
  
on main.tf line 1:  
1: resource "local_file" "my-pet" {  
  
Resource local_file.my-pet has  
lifecycle.prevent_destroy set, but the plan calls  
for this resource to be destroyed. To avoid this error  
and continue with the plan, either disable  
lifecycle.prevent_destroy or reduce the scope of the  
plan using the -target flag.
```



# ignore\_changes

main.tf

```
resource "local_file" "pet" {  
    filename = "v3-pets.txt"  
    content = "We love pets!"  
    file_permission = "0700"  
  
    lifecycle {  
        ignore_changes = [  
            content  
        ]  
    }  
}
```

main.tf

```
resource "local_file" "pet" {  
    filename = "v3-pets.txt"  
    content = "We love dogs!"  
    file_permission = "0700"  
  
    lifecycle {  
        ignore_changes = [  
            content  
        ]  
    }  
}
```

>\_

```
$ terraform apply  
local_file.pet: Refreshing state... [id=i-  
05cd83b221911acd5]  
  
Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
```



WECAMP

Seyed Saleh Miri

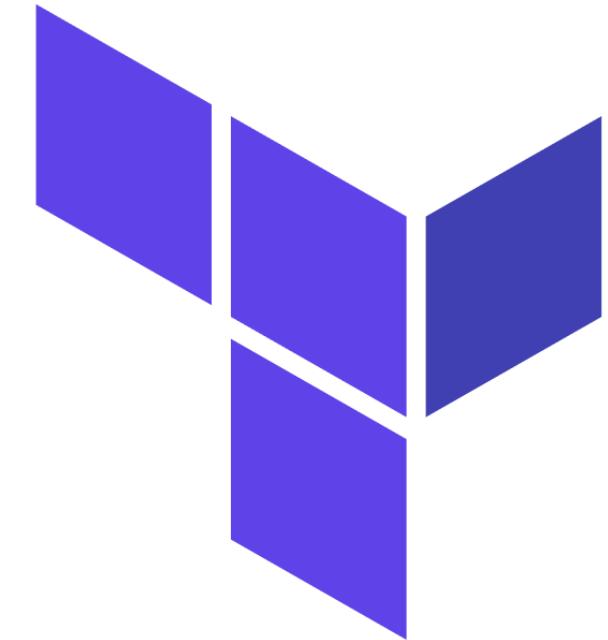
# 6<sup>th</sup> Session

[\(click here\)](#)



# 05 : Terraform Variables

- **Terraform Input Variables**
- **Demo**
- **Terraform Output Values**
- **Demo**
- **Terraform Local Values**
- **DRY**
- **Demo**



# Terraform Variables

Terraform  
Input  
Variables

Terraform  
Output  
Values

Terraform  
Local  
Values

# Terraform Input Variables

Input variables serve as **parameters** for a Terraform module, allowing aspects of the module to be **customized** without **altering** the module's own source code, and allowing modules to be **shared** between different configurations.

## Input Variables - Basics

1

Provide Input Variables **when prompted** during **terraform plan** or **apply**

2

Override **default variable values** using CLI argument **-var**

3

Override **default variable values** using **Environment Variables** (`TF_var_aa`)

4

Provide Input Variables using `terraform.tfvars` files

5

Provide Input Variables using `<any-name>.tfvars` file with CLI argument **-var-file**

7

Provide Input Variables using `auto.tfvars` files

8

Implement complex type **constructors** like **List** & **Map** in Input Variables

9

Implement **Custom Validation Rules** in Variables

10

Protect **Sensitive Input Variables**

## Terraform Input Variables



# Sample Main File

main.tf

```
resource "local_file" "pet" {
    filename = "/root/pets.txt"
    content = "We love pets!"

}

resource "random_pet" "my-pet" {
    prefix = "Mrs"
    separator = "."
    length = "1"
}
```

variables.tf

```
variable "filename" {

}
variable "content" {

}
variable "prefix" {

}
variable "separator" {

}
variable "length" {

}
```

# Variable Structure

main.tf

```
resource "local_file" "pet" {  
    filename = var.filename  
    content = var.content  
}  
  
resource "random_pet" "my-pet" {  
    prefix = var.prefix  
    separator = var.separator  
    length = var.length  
}
```

variables.tf

```
variable "filename" {  
}  
variable "content" {  
}  
variable "prefix" {  
}  
variable "separator" {  
}  
variable "length" {  
}
```



WECAMP

Seyed Saleh Miri

# Interactive Mode

```
>_
$ terraform apply
var.content
Enter a value: We love Pets!

var.filename
Enter a value: /root/pets.txt

var.length
Enter a value: 2

var.prefix
Enter a value: Mrs.

var.separator
Enter a value: .
```

# Environment Variables

```
>_
$ export TF_VAR_filename="/root/pets.txt"
$ export TF_VAR_content="We love pets!"
$ export TF_VAR_prefix="Mrs"
$ export TF_VAR_separator=". "
$ export TF_VAR_length="2"
$ terraform apply
```

# Command Line Flags

```
>_
```

```
$ terraform apply -var "filename=/root/pets.txt" -var "content=We love  
Pets!" -var "prefix=Mrs" -var "separator=." -var "length=2"
```

### main.tf

```
resource "local_file" "pet" {  
    filename = var.filename  
    content = var.content  
}  
  
resource "random_pet" "my-pet" {  
    prefix = var.prefix  
    separator = var.separator  
    length = var.length  
}
```

### variables.tf

```
variable "filename" {  
    default = "/root/pets.txt"  
}  
variable "content" {  
    default = "We love pets!"  
}  
variable "prefix" {  
    default = "Mrs"  
}  
variable "separator" {  
    default = "."  
}  
variable "length" {  
    default = 2  
}
```

## main.tf

```
resource "local_file" "pet" {
  filename = var.filename
  content = var.content
}

resource "random_pet" "my-pet" {
  prefix = var.prefix
  separator = var.separator
  length = var.length
}
```

```
variable "filename" {
  default = "/root/pets.txt"
  type = string
  description = "the path of local file"
  sensitive = true
}

variable "content" {
  default = "I love pets!"
  type = string
  description = "the content of the file"
}

variable "prefix" {
  default = "Mrs"
  type = string
  description = "the prefix to be set"
}

variable "separator" {
  default = "."
  type = string
  description = "the separator to be used"
```

# Variable Definition Files

```
terraform.tfvars
```

```
filename = "/root/pets.txt"
content = "We love pets!"
prefix = "Mrs"
separator = "."
length = "2"
```

```
>_
```

```
$ terraform apply -var-file variables.tfvars
```

terraform.tfvars

terraform.tfvars.json

\*.auto.tfvars

\*.auto.tfvars.json

Automatically Loaded



WECAMP

Seyed Saleh Miri

# 7<sup>th</sup> Session

[\(click here\)](#)



# Terraform Variables – Output Values

Output values are like the **return values** of a Terraform module and have several uses

1

A root module can use outputs to **print** certain values in the **CLI** output after running **terraform apply**.

2

A child module can use outputs to **expose a subset** of its resource attributes to a **parent module**.

3

When using **remote state**, root module outputs can be accessed by other configurations via a **terraform\_remote\_state** data source.

Advanced



# Create Output Variables

```
# Define output variables
output "virtual_machine_name" {
    description = "The name of the virtual machine"
    value       = vsphere_virtual_machine.saleh_vm.name
}

output "vm_state" {
    description = "The current state of the VM"
    value       = vsphere_virtual_machine.saleh_vm.power_state
}

output "vm_id" {
    description = "The ID of the VM"
    value       = vsphere_virtual_machine.saleh_vm.id
}
```

The code defines three output variables. Each output block consists of a label, a description, and a value. The value is derived from a resource labeled 'vsphere\_virtual\_machine' and a specific attribute of that resource.

Resource Label 1	Resource Label 2	Attributes
virtual_machine_name	vm_state	power_state
vm_id		

# Terraform Variables - Output Values

```
Plan: 1 to add, 0 to change, 0 to destroy.

Changes to Outputs:
+ virtual_machine_name = "saleh-vm"
+ vm_id                = (known after apply)
+ vm_state              = (known after apply)

Do you want to perform these actions?
Terraform will perform the actions described above.
Only 'yes' will be accepted to approve.

Enter a value: yes

vsphere_virtual_machine.saleh_vm: Creating...
vsphere_virtual_machine.saleh_vm: Creation complete after 8s

Apply complete! Resources: 1 added, 0 changed, 0 destroyed.

Outputs:

virtual_machine_name = "saleh-vm"
vm_id = "4233465a-f69d-332a-c3e1-7fd11c29f9c3"
vm_state = "on"
```



# File Resource and Outputs example

```
resource "local_file" "output_file" {
  filename = "./${var.vm_name}/output.txt"
  content  = <<EOF
The VM is ${vsphere_virtual_machine.saleh_vm.power_state}
EOF
}
```

# Terraform Output

main.tf

```
resource "local_file" "pet" {
    filename      = "pets.txt"
    content       = "We love pets!"
    file_permission = "0777"
}
resource "random_pet" "cat" {
    length      = "2"
    separator   = "-"
}
output content {
    value        = local_file.pet.content
    sensitive    = false
    description  = "Print the content of the file"
}
output pet-name {
    value        = random_pet.cat.id
    sensitive    = false
    description  = "Print the name of the pet"
}
```

>\_

```
$ terraform output
content = We love pets!
pet-name = huge-owl
```

```
$ terraform output pet-name
pet-name = huge-owl
```

# 8<sup>th</sup> Session

[\(click here\)](#)



# Terraform Variables – Local Values

A local value assigns a name to an expression, so you can use that name multiple times within a module without repeating it.

Local values are like a function's temporary local variables.

Once a local value is declared, you can reference it in expressions as `local.<NAME>`.

Local values can be helpful to avoid repeating the same values or expressions multiple times in a configuration

If overused they can also make a configuration hard to read by future maintainers by hiding the actual values used

The ability to easily change the value in a central place is the key advantage of local values.

In short, Use local values only in moderation

```
# Local Values Block
locals {
    vm_name = "${var.env}-${var.project_name}-vm"
}

# Resource Block
resource "vsphere_virtual_machine" "saleh_vm" {
    name      = local.vm_name
    num_cpus = var.vm_cpu
    memory   = var.vm_ram
```



# Don't Repeat Yourself

DRY



Before DRY

```
x = 10;  
print(10);  
y = 10 + 20;
```



After DRY

```
value = 10;  
print(value);  
y = value + 20;
```

DRY

افزایش خوانایی کد  
کاهش احتمال خطا  
تسهیل در نگهداری و تغییرات آینده  
کاهش زمان توسعه



WECAMP

Seyed Saleh Miri

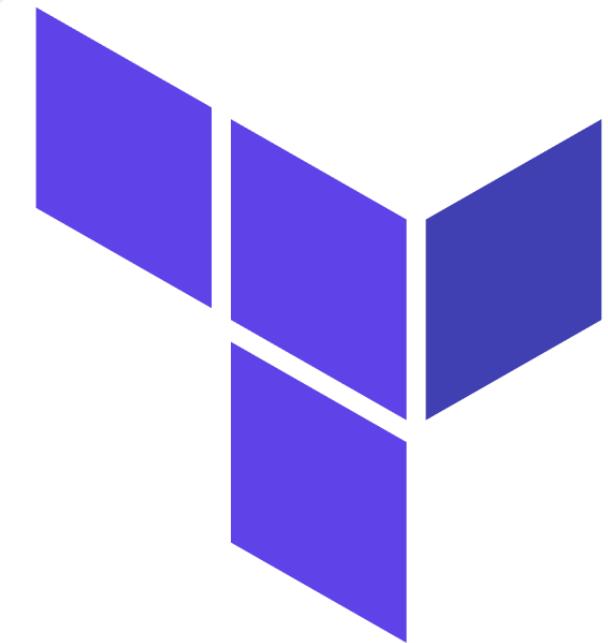
# 9<sup>th</sup> Session

[\(click here\)](#)



# 06 : Terraform Data Sources

- **Terraform Data Source**
- **Demo**



# Terraform Datasources

Data sources allow data to be fetched or computed for use elsewhere in Terraform configuration.

Use of data sources allows a Terraform configuration to make use of information defined outside of Terraform, or defined by another separate Terraform configuration.

A data source is accessed via a special kind of resource known as a *data resource*, declared using a *data block*

Each data resource is associated with a *single data source*, which determines the *kind of object (or objects)* it reads and what *query constraint arguments* are available

Data resources have the *same dependency resolution behavior* as defined for managed resources. Setting the *depends\_on* meta-argument within data blocks *defers* reading of the data source until after all changes to the dependencies have been *applied*.

```
# Data Block
data "vsphere_datacenter" "dc" {
    name = "Datacenter-lab"
}

data "vsphere_compute_cluster" "cluster" {
    name          = "vSAN-lab"
    datacenter_id = data.vsphere_datacenter.dc.id
}

data "vsphere_datastore" "ds" {
    name          = "vsanDatastore"
    datacenter_id = data.vsphere_datacenter.dc.id
}

data "vsphere_network" "network" {
    name          = "VM Network"
    datacenter_id = data.vsphere_datacenter.dc.id
}
```

# Terraform Datasources

We can refer the data resource in a resource as depicted

## Meta-Arguments for Datasources

```
# Resource Block
resource "vsphere_virtual_machine" "saleh_vm" {
    name      = var.vm_name
    num_cpus  = var.vm_cpu
    memory    = var.vm_ram

    resource_pool_id = data.vsphere_compute_cluster.
    datastore_id     = data.vsphere_datastore.ds.id
    guest_id         = "otherGuest"

    network_interface {
        network_id  = data.vsphere_network.network.id
        adapter_type = "vmxnet3"
    }
}
```

Data resources support the **provider** meta-argument as defined for managed resources, with the **same syntax and behavior**.

Data resources **do not currently have** any customization settings available for their **lifecycle**, but the **lifecycle** nested block is **reserved** in case any are added in future versions.

Data resources support **count** and **for\_each** meta-arguments as defined for managed resources, with the **same syntax and behavior**. Each instance will **separately** read from its data source with its own variant of the constraint arguments, producing an **indexed result**.



```
>_
$ cat /root/dog.txt
Dogs are awesome!
```

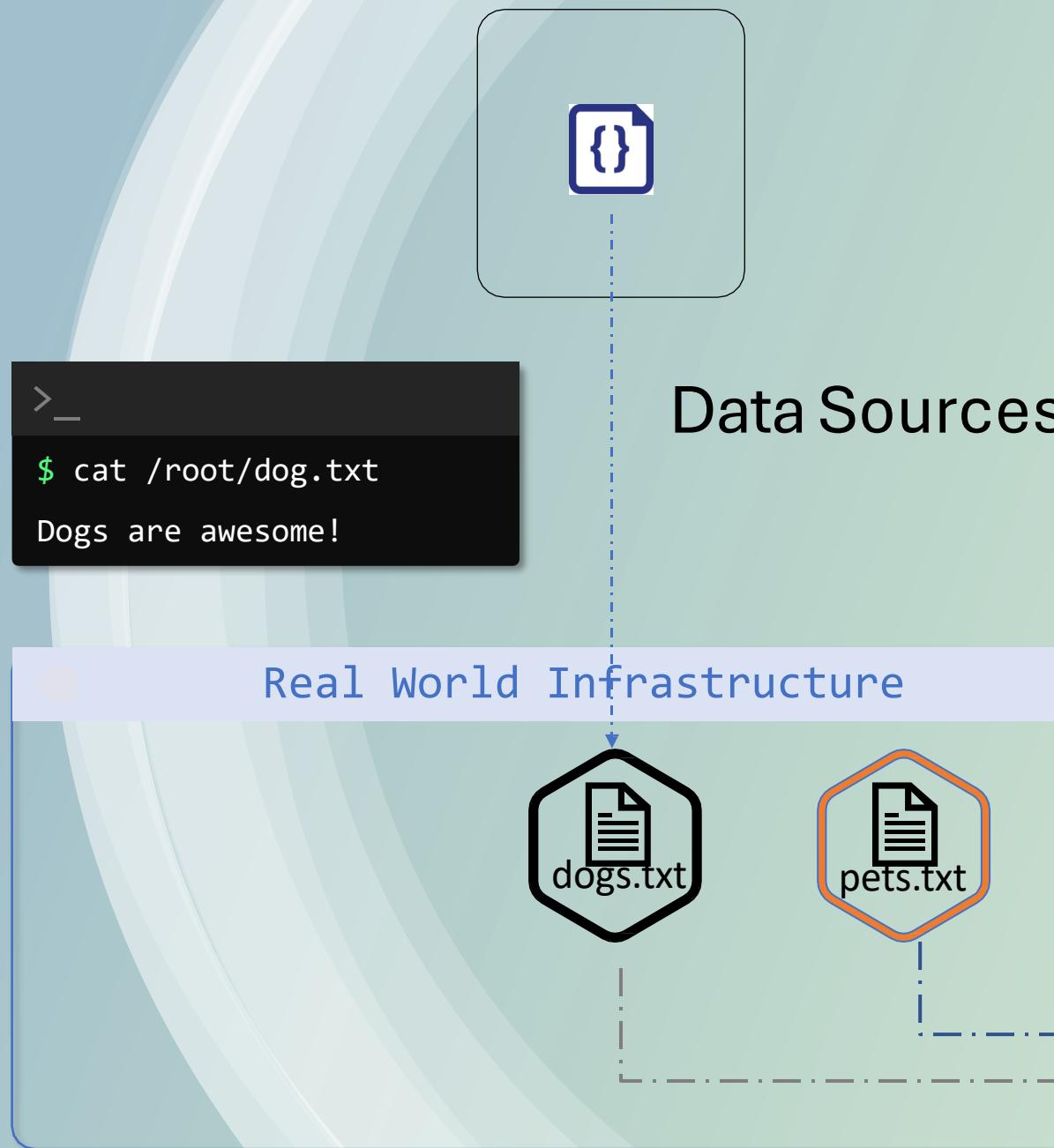


```
main.tf
```

```
resource "local_file" "pet" {
  filename = "/root/pets.txt"
  content = "We love pets!"
}
```

```
terraform.tfstate
```

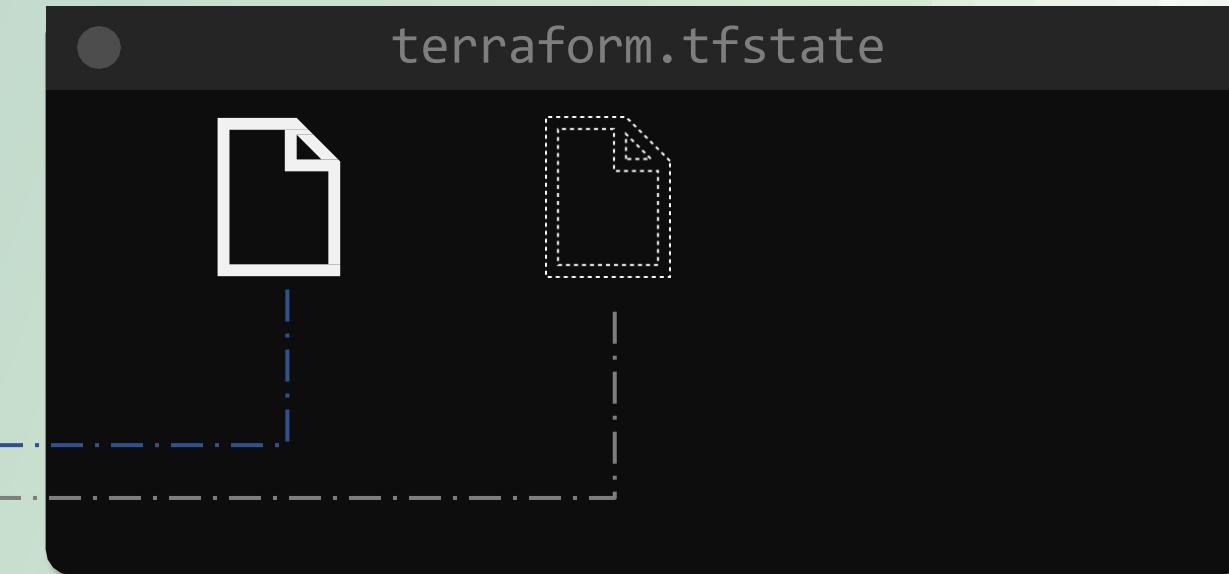
The diagram shows a single white file icon with a black outline, representing the state of files in the `terraform.tfstate` file. A dashed arrow points from this icon down to a box labeled 'terrafrom.tfstate'. This indicates that only the 'dogs.txt' file is currently tracked by Terraform's state file.



main.tf

```
resource "local_file" "pet" {
  filename = "/root/pets.txt"
  content = data.local_file.dog.content
}

data "local_file" "dog" {
  filename = "/root/dog.txt"
}
```



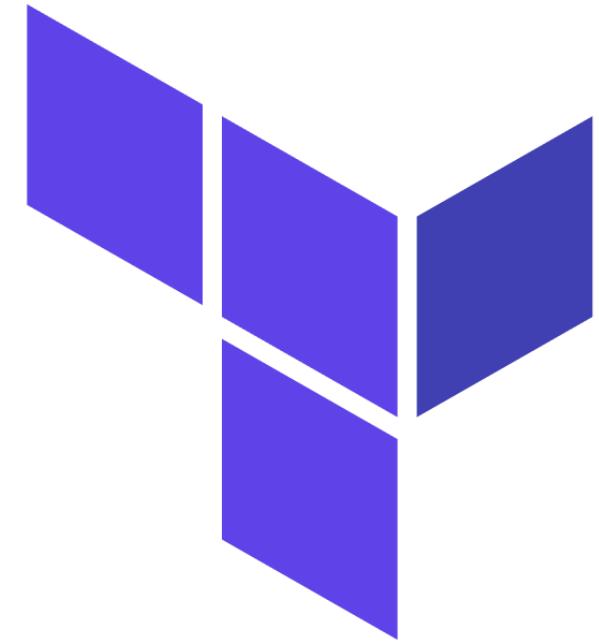
# 10<sup>th</sup> Session

[\(click here\)](#)



# 07 : Terraform Provisioner

- **Terraform Provisioners**
- **Types of Provisioners**
- **Connection Block**
- **File Provisioner**
- **local-exec Provisioner**
- **remote-exec Provisioner**
- **Null-Resource & Provisioners**
- **Demo**



# Terraform Provisioners

Provisioners can be used to model **specific actions** on the **local machine** or on a **remote machine** in order to **prepare servers**

Passing **data** into virtual machines and other compute resources

Running **configuration management** software (packer, chef, ansible)

**Creation-Time Provisioners**

**Failure Behaviour:** Continue: Ignore the error and continue with **creation** or **destruction**.

Provisioners are a **Last Resort**

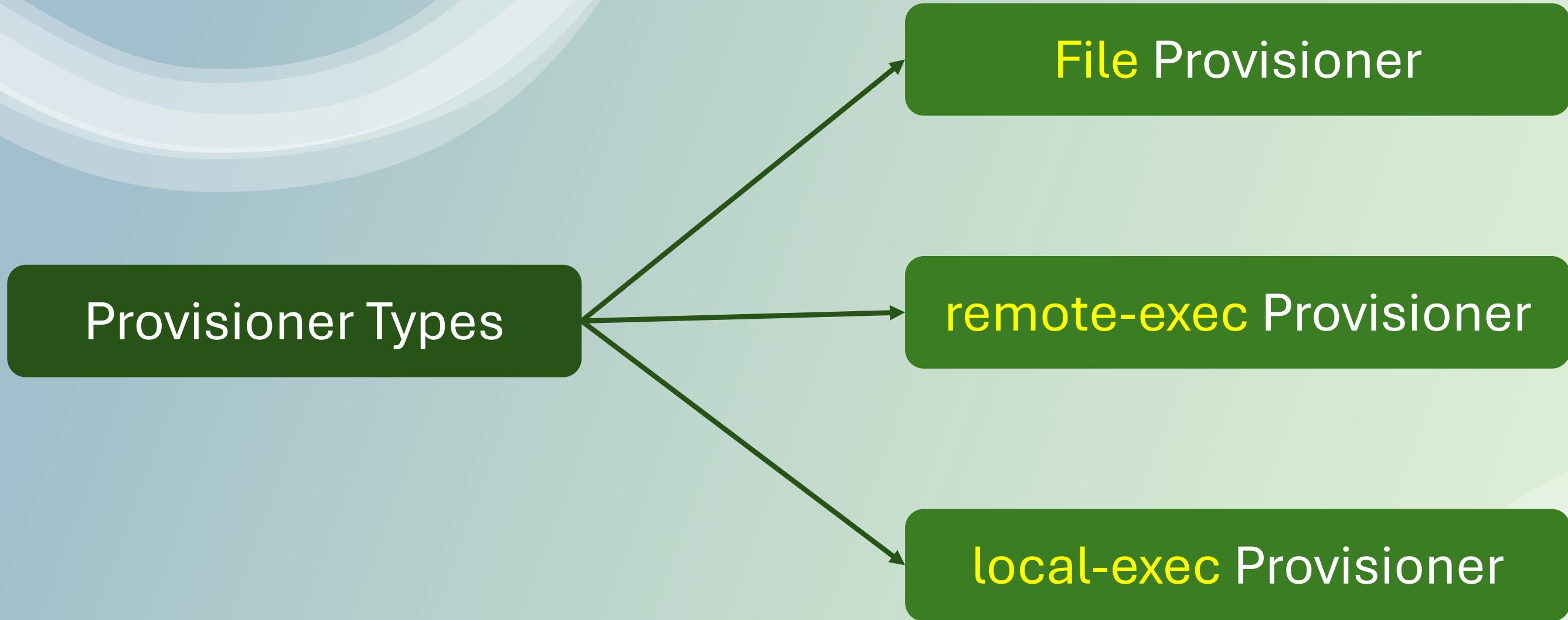
**First-class Terraform provider functionality** may be available

**Destroy-Time Provisioners**

**Failure Behaviour:** Fail: Raise an error and stop applying (the default behavior). If creation provisioner, **taint** resource



# Types of Provisioners



# Connection Block

Most provisioners require access to the remote resource via **SSH** or **WinRM**, and expect a nested **connection block** with details about how to **connect**.

Expressions in connection blocks **cannot** refer to their parent resource by name. Instead, they can use the special **self** object.

```
connection {  
    type      = "ssh"  
    user      = var.ssh_user  
    password  = var.ssh_pass  
    host      = self.default_ip_address  
}
```

```
connection {  
    type      = "winrm"  
    user      = "administrator"  
    password  = "Adm!n!stR@t0r"  
    host      = self.default_ip_address  
    https     = false  
    port      = 5985  
    insecure  = true  
}
```



# File Provisioner

## File Provisioner

- File Provisioner is used to copy files or directories from the machine executing Terraform to the newly created resource.
- The file provisioner supports both ssh and winrm type of connections

```
provisioner "file" {
  source      = "scripts/hello.sh"
  destination = "/tmp/hello.sh"

  connection {
    type        = "ssh"
    user        = var.ssh_user
    password    = var.ssh_pass
    host        = self.default_ip_address
  }
}
```

```
provisioner "file" {
  source      = "scripts/hello.sh"
  destination = "/tmp/hello.sh"
}

connection {
  type        = "ssh"
  user        = var.ssh_user
  password    = var.ssh_pass
  host        = self.default_ip_address
}
```

# remote-exec Provisioner

## remote-exec Provisioner

- The remote-exec provisioner invokes a script on a remote resource after it is created.
- This can be used to run a configuration management tool, bootstrap into a cluster, etc.

```
provisioner "remote-exec" {
  inline = [
    "chmod +x /tmp/hello.sh",
    "bash /tmp/hello.sh"
  ]

  connection {
    type      = "ssh"
    user      = var.ssh_user
    password  = var.ssh_pass
    host      = self.default_ip_address
  }
}
```



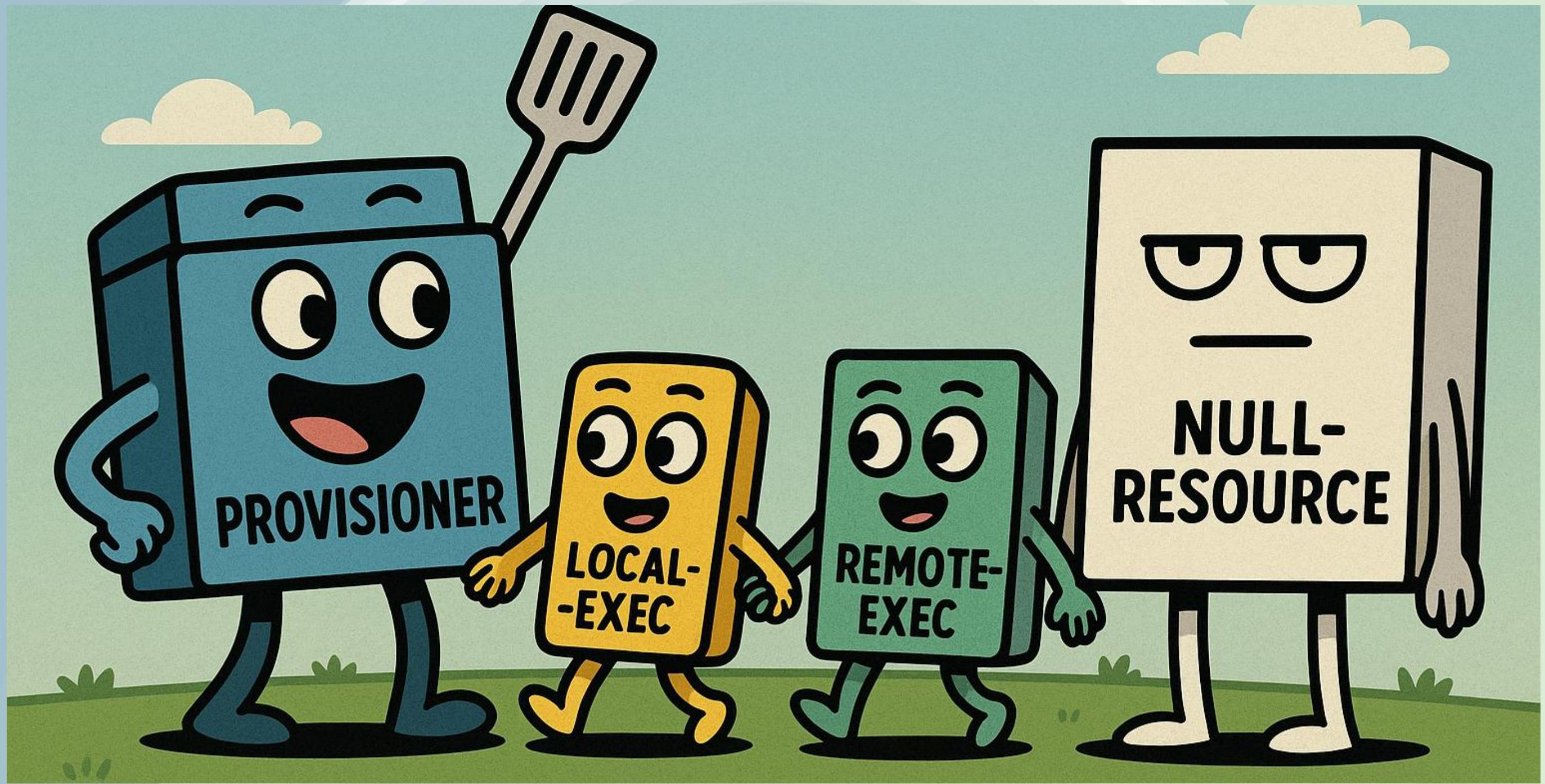
# local-exec Provisioner

## local-exec Provisioner

- The local-exec provisioner invokes a local executable after a resource is created.
- This invokes a process on the machine running Terraform, not on the resource.

```
provisioner "local-exec" {
  command = "echo 'VM ${self.name} created on Today!' >> ./logs/vm_creation.log && false"
}
```





## NULL Provider & NULL Resource

# Null-Resource & Provisioners

null provider  
null\_resource

- If you need to run provisioners that aren't directly associated with a specific resource, you can associate them with a `null_resource`.
- Instances of `null_resource` are treated like normal resources, but they don't do anything.
- Same as other resource, you can configure provisioners and connection details on a `null_resource`.

```
required_providers {  
    vsphere = {  
        source  = "hashicorp/vsphere"  
        version = "2.8.1"  
    }  
    local = {  
        source  = "hashicorp/local"  
        version = "2.5.1"  
    }  
    null = {  
        source  = "hashicorp/null"  
        version = "3.2.2"  
    }  
}
```

```
resource "null_resource" "log_vm_creation" {  
    provisioner "local-exec" {  
        command      = "echo 'VM ${var.vm_name} created on Today!' >> ./logs/vm_creation.log"  
        interpreter = ["/bin/bash", "-c"]  
    }  
  
    depends_on = [  
        vsphere_virtual_machine.this  
    ]  
}
```

# 11<sup>th</sup> Session

[\(click here\)](#)



# Root Module and Child Module

```
v02-terraform-manifests-Module
  modules\vcenter_vm
    main.tf
    outputs.tf
    variables.tf
  c1-versions.tf
  c2-variables.tf
  c3-main.tf
  c4-datasource.tf
  c5-output.tf
  terraform.tfvars
  README.md
```

Child Module

Root Module



# Root Module vs Child Module

Child Module	Root Module	Characteristic
No ✘	Yes (with <code>terraform apply</code> ) ✓	Direct execution
Yes – must be called by another module ✓	No ✘	Requires invocation from elsewhere
No – only through a parent module ✘	Yes ✓	Executed independently
No – usually does not include provider ✘	Yes ✓	Typically includes provider

# Terraform Modules

Modules are **containers** for multiple resources that are used together. A module consists of a collection of **.tf** files kept together in a directory.

Modules are the main way to **package** and reuse resource configurations with Terraform.

Every Terraform configuration has at least one module, known as its **root module**, which consists of the resources defined in the **.tf** files in the **main working directory**.

A Terraform module (usually the **root module** of a configuration) can call **other modules** to include their resources into the configuration.

A module that has been **called** by another module is often referred to as a **child module**.

Child modules **can be called multiple times** within the same configuration, and **multiple configurations** can use the same child module.

In addition to modules from **the local filesystem**, Terraform can load modules from a **public or private registry**.

This makes it possible to **publish modules** for others to **use**, and to use modules that others have published.

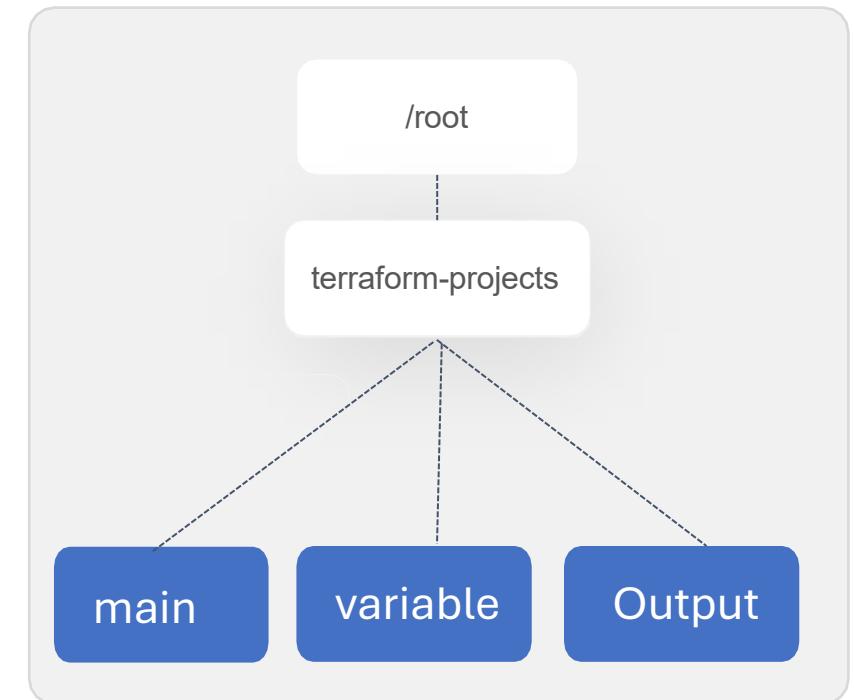


```
>_
$ ls /root/terraform-projects/v01-terraform-playground-Simple
main.tf    variables.tf    output.tf
```

```
main.tf

resource "local_file" "pet" {
  filename = var.filename
  content  = var.content
}

resource "random_pet" "my-pet" {
  prefix   = var.prefix
  separator = var.separator
  length    = var.length
}
```



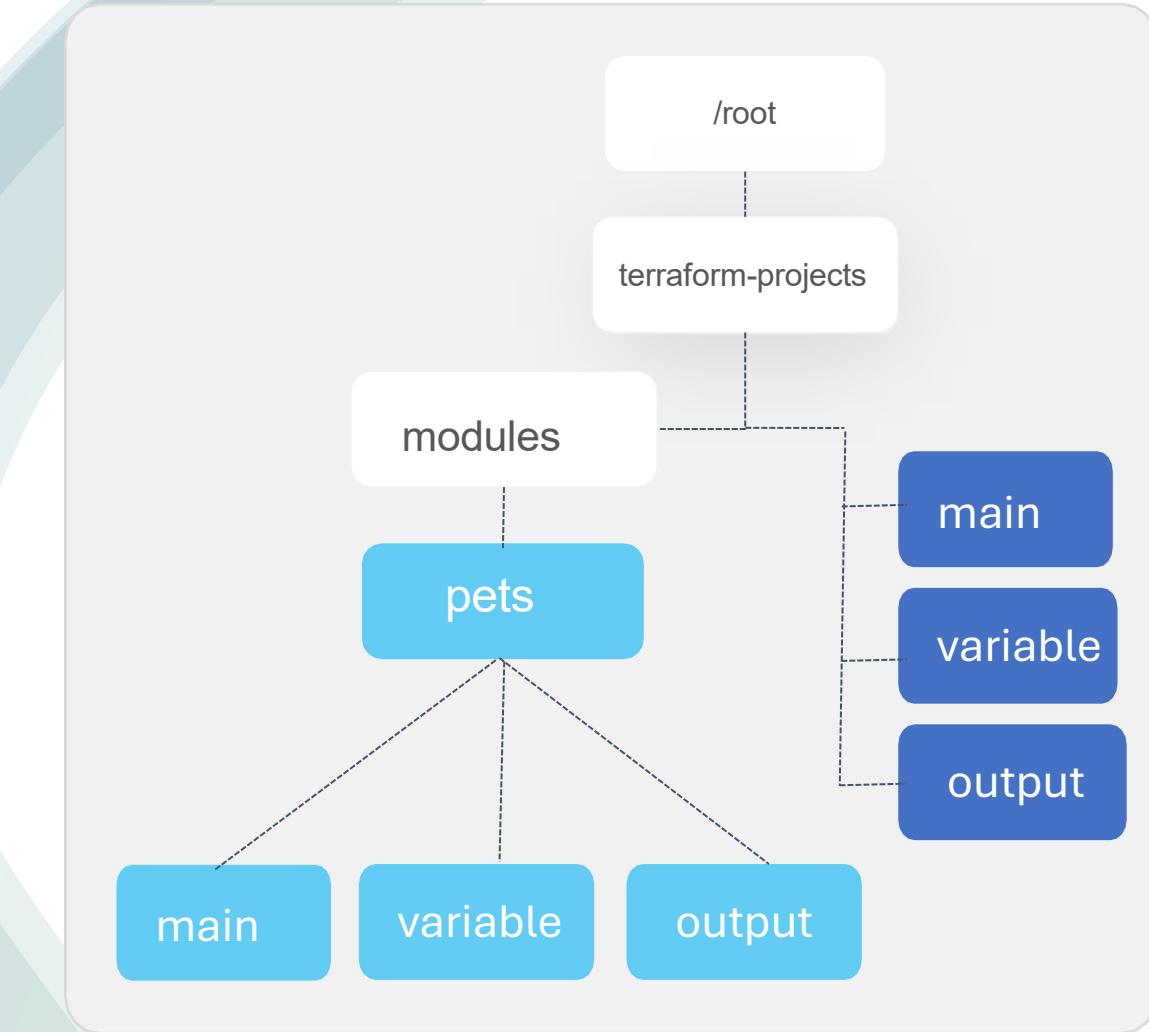
```
modules/pets/main.tf

resource "local_file" "pet" {
    filename = var.filename
    content  = var.content
}

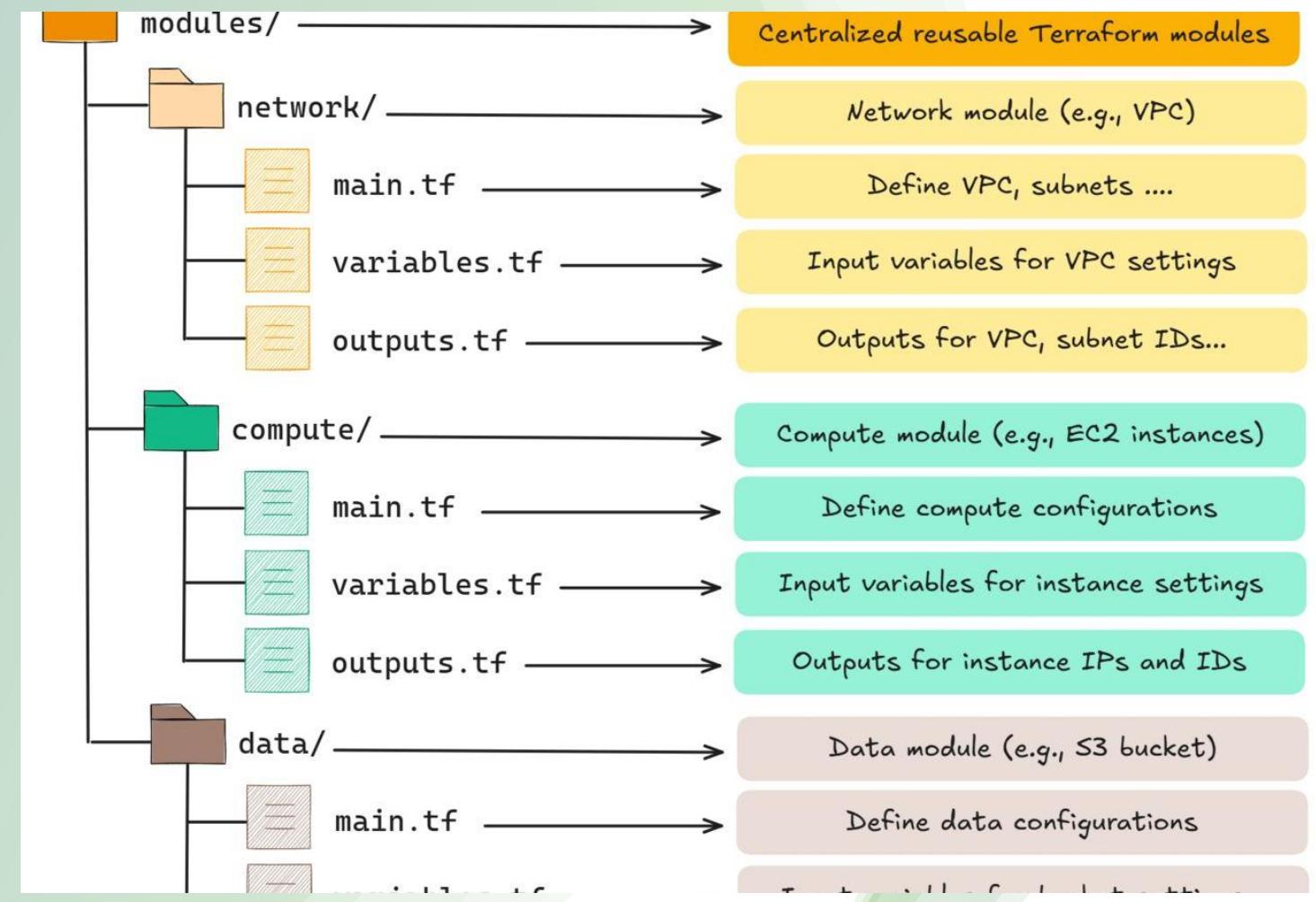
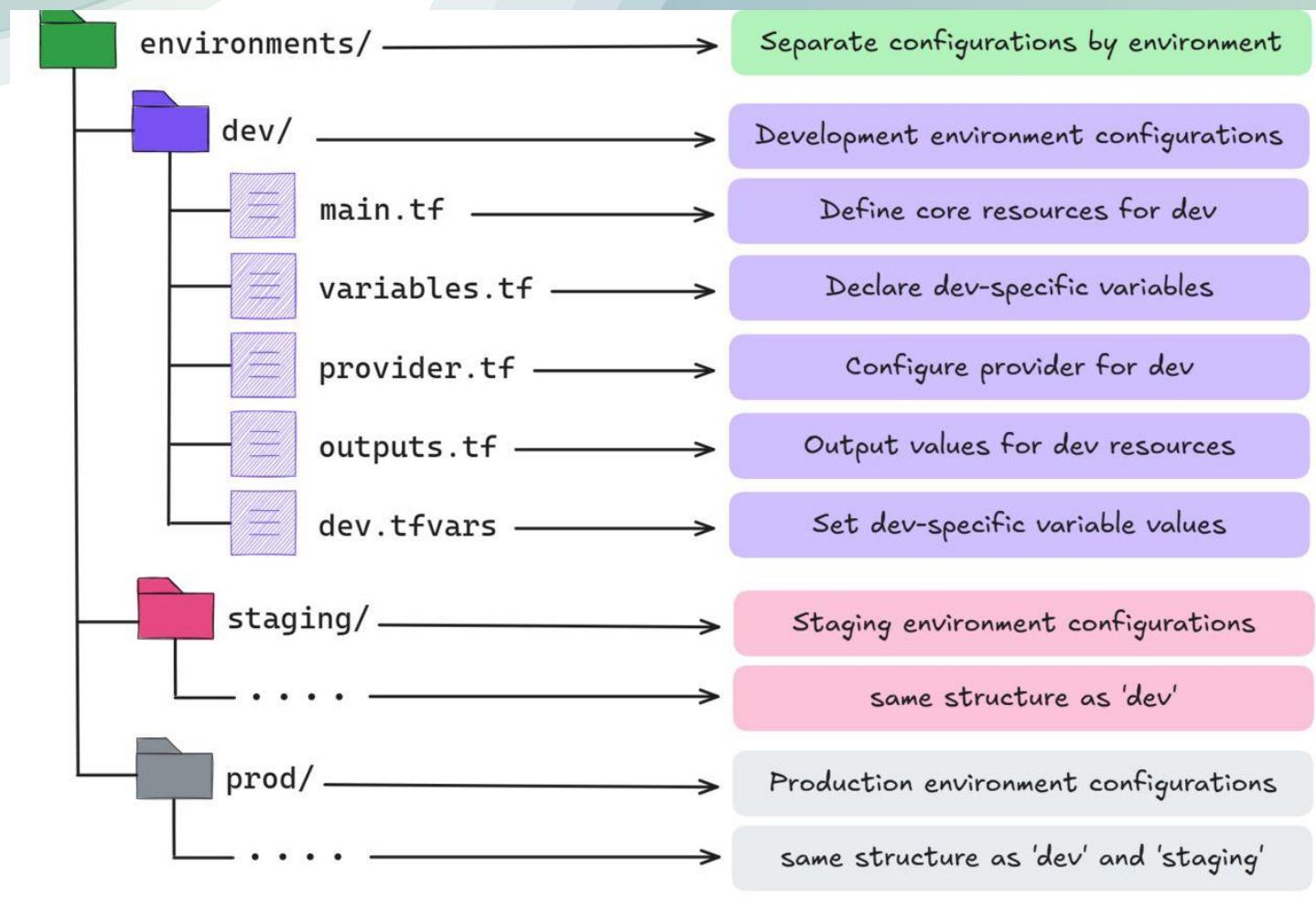
resource "random_pet" "my-pet" {
    prefix    = var.prefix
    separator = var.separator
    length    = var.length
}
```

```
main.tf

module "pets_module" {
    source      = "./modules/pets"
    filename    = var.filename
    content     = var.content
    prefix      = var.prefix
    separator   = var.separator
    length      = var.length
}
```



# Terraform Module Structure



# Terraform Registry: Publicly Available Modules

# Terraform Registry

What is the  
Terraform  
Registry?

Why Use the  
Terraform  
Registry?

Types of  
Available  
Modules

Terraform  
Automatically  
Downloads  
Modules

Standard Module  
Structure in  
the Registry

Searching and  
Choosing the  
Right Module

Verified and  
Certified  
Modules

Reusability  
Across  
Projects



```
# Module Block

module "saleh_vm" {
    source          = "modules/vcenter_vm"
    vm_name        = var.vm_name
    num_cpus       = var.num_cpus
    memory         = var.memory
    resource_pool_id = data.vsphere_compute_cluster.cluster.resource_pool_id
    datastore_id    = data.vsphere_datastore.ds.id
    guest_id        = "otherGuest"
    network_id      = data.vsphere_network.network.id
    adapter_type    = "vmxnet3"
    vm_size         = var.vm_size
    eagerly_scrub   = var.eagerly_scrub
    thin_provisioned = var.thin_provisioned
}
```

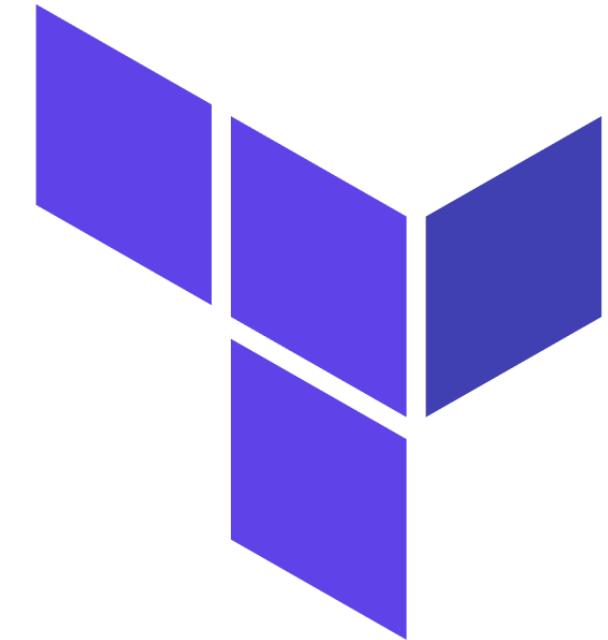
# 12<sup>th</sup> Session

[\(click here\)](#)



# 09 : Terraform Commands

- **Terraform Validate**
- **Terraform Format**
- **Terraform Show**
- **Terraform Providers**
- **Terraform Output**
- **Terraform Refresh**
- **Terraform Graph**
- **Demo**



# Terraform validate

main.tf

```
resource "local_file" "pet" {  
    filename = "/root/pets.txt"  
    content = "We love pets!"  
    file_permissions = "0700"  
}
```

>\_

```
$ terraform validate  
Success! The configuration is valid.  
  
$ terraform validate  
  
Error: Unsupported argument  
  
on main.tf line 4, in resource "local_file" "pet":  
  4:   file_permissions = "0777"  
  
An argument named "file_permissions" is not expected  
here. Did you mean "file_permission"?
```

# Terraform format

main.tf

```
resource "local_file" "pet" {  
    filename      = "/root/pets.txt"  
    content       = "We love pets!"  
    file_permission = "0700"  
}
```

>\_

```
$ terraform fmt  
main.tf
```

# Terraform show

```
>_
```

```
$ terraform show  
# local_file.pet:  
resource "local_file" "pet" {  
    content          = "We love pets!"  
    directory_permission = "0777"  
    file_permission     = "0777"  
    filename           = "/root/pets.txt"  
    id                 =  
"cba595b7d9f94ba1107a46f3f731912d95fb3d2c"  
}
```

```
>_
```

```
$ terraform show -json  
{"format_version":"0.1","terraform_version":"0.13.0",  
"values": {"root_module": {"resources": [{"address":  
"local_file.pet", "mode": "managed", "type": "local_file",  
"name": "pet", "provider_name": "registry.terraform.io/hashicorp/local",  
"schema_version": 0, "values": {"content": "We love  
pets!", "content_base64": null, "directory_permission":  
"0777", "file_permission": "0777", "filename": "/root/  
pets.txt", "id": "cba595b7d9f94ba1107a46f3f731912d95f  
b3d2c", "sensitive_content": null}}]}}}
```

# Terraform providers

main.tf

```
resource "local_file" "pet" {  
    filename      = "/root/pets.txt"  
    content       = "We love pets!"  
    file_permission = "0700"  
}
```

>\_

```
$ terraform providers  
Providers required by configuration:  
.└ provider[registry.terraform.io/hashicorp/local]
```

Providers required by state:

```
provider[registry.terraform.io/hashicorp/local]
```

```
$ terraform providers mirror /root/terraform/new_local_file
```

- Mirroring hashicorp/local...
  - Selected v1.4.0 with no constraints
  - Downloading package for windows\_amd64...
  - Package authenticated: signed by HashiCorp

# Terraform output

main.tf

```
resource "local_file" "pet" {
    filename      = "/root/pets.txt"
    content       = "We love pets!"
    file_permission = "0777"
}
resource "random_pet" "cat" {
    length      = "2"
    separator   = "-"
}
output content {
    value        = local_file.pet.content
    sensitive    = false
    description  = "Print the content of the file"
}
output pet-name {
    value        = random_pet.cat.id
    sensitive    = false
    description  = "Print the name of the pet"
}
```

>\_

```
$ terraform output
content = We love pets!
pet-name = huge-owl
```

```
$ terraform output pet-name
pet-name = huge-owl
```

# Terraform refresh

main.tf

```
resource "local_file" "pet" {  
    filename      = "/root/pets.txt"  
    content       = "We love pets!"  
    file_permission = "0777"  
}  
resource "random_pet" "cat" {  
    length      = "2"  
    separator   = "-"  
}
```

>\_

```
$ terraform refresh  
random_pet.cat: Refreshing state... [id=huge-owl]  
local_file.pet: Refreshing state...  
[id=cba595b7d9f94ba1107a46f3f731912d95fb3d2c]
```

```
$ terraform plan
```

Refreshing Terraform state in-memory prior to plan...  
The refreshed state will be used to calculate this  
plan, but will not be  
persisted to local or remote state storage.

```
random_pet.cat: Refreshing state... [id=huge-owl]  
local_file.pet: Refreshing state...  
[id=cba595b7d9f94ba1107a46f3f731912d95fb3d2c]  
-----
```

No changes. Infrastructure is up-to-date.



WECAMP

Seyed Saleh Miri

# Terraform Graph

main.tf

```
resource "local_file" "pet" {
  filename = "/root/pets.txt"
  content  = "My favorite pet is ${random_pet.my-pet.id}"
}
resource "random_pet" "my-pet" {
  prefix = "Mr"
  separator = "."
  length = "1"
}
```

>\_

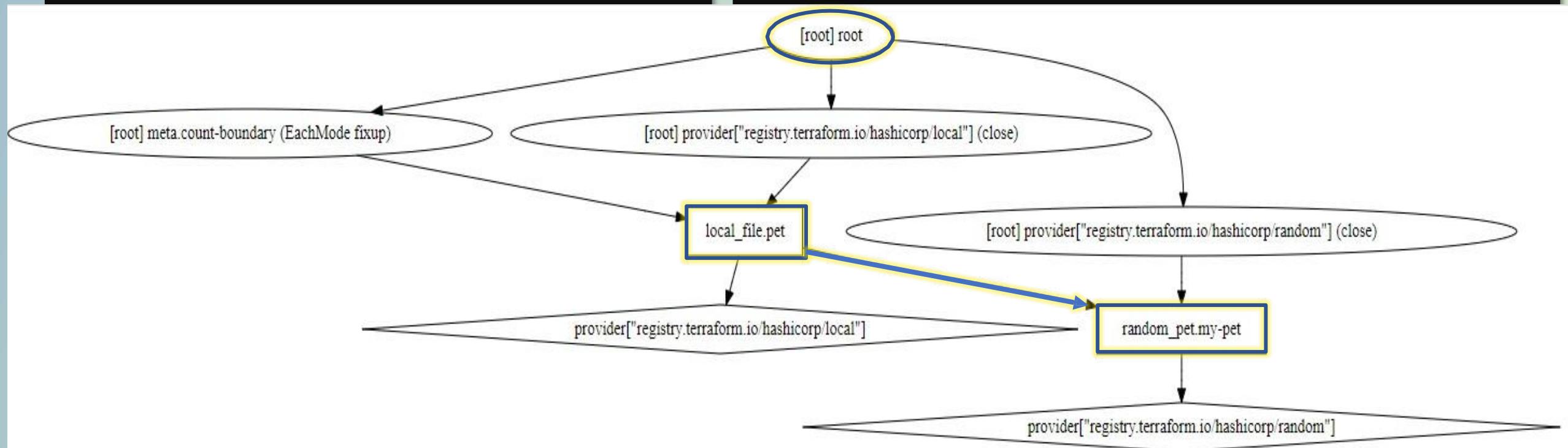
```
$ terraform graph
digraph {
  compound = "true"
  newrank = "true"
  subgraph "root" {
    "[root] local_file.pet (expand)" [label =
"local_file.pet", shape = "box"]
    "[root]
provider[\"registry.terraform.io/hashicorp/local\"]" [label =
"provider[\"registry.terraform.io/hashicorp/local\"], shape =
"diamond"]
    "[root]
provider[\"registry.terraform.io/hashicorp/random\"]" [label =
"provider[\"registry.terraform.io/hashicorp/random\"], shape =
"diamond"]
    "[root] random_pet.my-pet (expand)" [label =
"random_pet.my-pet", shape = "box"]
    "[root] local_file.pet (expand)" -> "[root]
provider[\"registry.terraform.io/hashicorp/local\"]"
    "[root] local_file.pet (expand)" -> "[root]
random_pet.my-pet (expand)"
    "[root] meta.count-boundary (EachMode fixup)" -
> "[root] local_file.pet (expand)"
    "[root]
provider[\"registry.terraform.io/hashicorp/local\"] (close)" ->
```

## main.tf

```
resource "local_file" "pet" {
  filename = "/root/pets.txt"
  content  = "My favorite pet is ${random_pet.my-pet.id}"
}
resource "random_pet" "my-pet" {
  prefix = "Mr"
  separator = "."
  length = "1"
}
```

>\_

```
$ apt update
$ apt install graphviz -y
$ terraform graph | dot -Tsvg > graph.svg
```



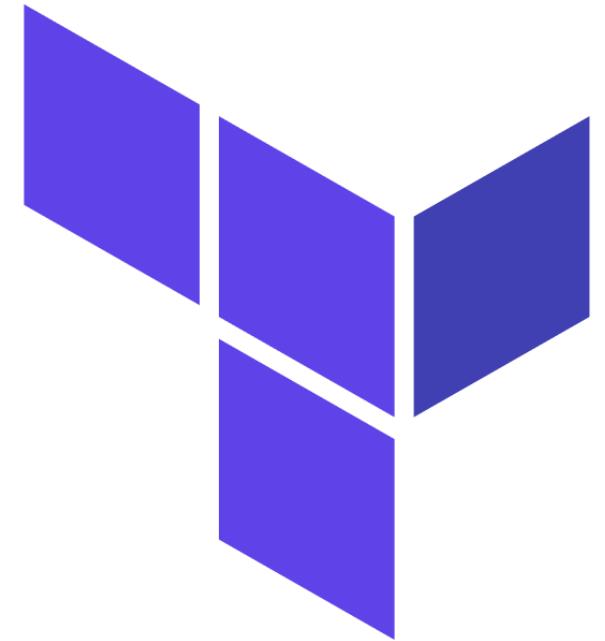
# 13<sup>th</sup> Session

[\(click here\)](#)



# 10 : Terraform Debug

- Log Levels
- Demo



# Log Levels

```
>_  
# export TF_LOG=<log_level>  
$ export TF_LOG=TRACE
```

INFO

WARNING

ERROR

DEBUG

TRACE

```
>_  
$ terraform plan  
----  
2020/10/18 22:08:30 [INFO] Terraform version: 0.13.0  
2020/10/18 22:08:30 [INFO] Go runtime version: go1.14.2  
2020/10/18 22:08:30 [INFO] CLI args: []string{"C:\\Windows\\system32\\terraform.exe", "plan"}  
2020/10/18 22:08:30 [DEBUG] Attempting to open CLI config file: C:\\Users\\vpala\\AppData\\Roaming\\terraform.rc  
2020/10/18 22:08:30 [DEBUG] File doesn't exist, but doesn't need to. Ignoring.  
2020/10/18 22:08:30 [DEBUG] ignoring non-existing provider search directory terraform.d/plugins  
2020/10/18 22:08:30 [DEBUG] ignoring non-existing provider search directory C:\\Users\\vpala\\AppData\\Roaming\\terraform.d\\plugins  
2020/10/18 22:08:30 [DEBUG] ignoring non-existing provider search directory  
C:\\Users\\vpala\\AppData\\Roaming\\HashiCorp\\Terraform\\plugins  
2020/10/18 22:08:30 [INFO] CLI command args: []string{"plan"}  
2020/10/18 22:08:30 [WARN] Log levels other than TRACE are currently unreliable, and are supported only for backward  
compatibility.  
  Use TF_LOG=TRACE to see Terraform's internal logs.  
----  
2020/10/18 22:08:30 [DEBUG] New state was assigned lineage "f413959c-538a-f9ce-524e-1615073518d4"  
2020/10/18 22:08:30 [DEBUG] checking for provisioner in "."  
2020/10/18 22:08:30 [DEBUG] checking for provisioner in "C:\\Windows\\system32"  
2020/10/18 22:08:30 [INFO] Failed to read plugin lock file .terraform\\plugins\\windows_amd64\\lock.json: open  
.terraform\\plugins\\windows_amd64\\lock.json: The system cannot find the path specified.  
2020/10/18 22:08:30 [INFO] backend/local: starting Plan operation  
2020-10-18T22:08:30.625-0400 [INFO] plugin: configuring client automatic mTLS  
2020-10-18T22:08:30.646-0400 [DEBUG] plugin: starting plugin:  
path=.terraform/plugins/registry.terraform.io/hashicorp/aws/3.11.0/windows_amd64/terraform-provider-aws_v3.11.0_x5.exe  
args=[.terraform/plugins/registry.terraform.io/hashicorp/aws/3.11.0/windows_amd64/terraform-provider-aws_v3.11.0_x5.exe]  
2020-10-18T22:08:30.935-0400 [DEBUG] plugin: plugin started:  
path=.terraform/plugins/registry.terraform.io/hashicorp/aws/3.11.0/windows_amd64/terraform-provider-aws_v3.11.0_x5.exe  
pid=34016  
2020-10-18T22:08:30.935-0400 [DEBUG] plugin: waiting for RPC address:  
path=.terraform/plugins/registry.terraform.io/hashicorp/aws/3.11.0/windows_amd64/terraform-provider-aws_v3.11.0_x5.exe  
2020-10-18T22:08:30.974-0400 [INFO] plugin.terraform-provider-aws_v3.11.0_x5.exe: configuring server automatic mTLS:
```

```
>_
```

```
$ export TF_LOG_PATH=/tmp/terraform.log
```

```
$ head -10 /tmp/terraform.logs
```

```
-----
2020/10/18 22:08:30 [INFO] Terraform version: 0.13.0
2020/10/18 22:08:30 [INFO] Go runtime version: go1.14.2
2020/10/18 22:08:30 [INFO] CLI args: []string{"C:\\Windows\\system32\\terraform.exe",
"plan"}
2020/10/18 22:08:30 [DEBUG] Attempting to open CLI config file:
C:\\Users\\vpala\\AppData\\Roaming\\terraform.rc
2020/10/18 22:08:30 [DEBUG] File doesn't exist, but doesn't need to. Ignoring.
2020/10/18 22:08:30 [DEBUG] ignoring non-existing provider search directory
terraform.d/plugins
2020/10/18 22:08:30 [DEBUG] ignoring non-existing provider search directory
C:\\Users\\vpala\\AppData\\Roaming\\terraform.d\\plugins
2020/10/18 22:08:30 [DEBUG] ignoring non-existing provider search directory
C:\\Users\\vpala\\AppData\\Roaming\\HashiCorp\\Terraform\\plugins
2020/10/18 22:08:30 [INFO] CLI command args: []string{"plan"}
```

```
$ unset TF_LOG_PATH
```



WECAMP

Seyed Saleh Miri

# 14<sup>th</sup> Session

[\(click here\)](#)

